Core Java Programming

Code-304

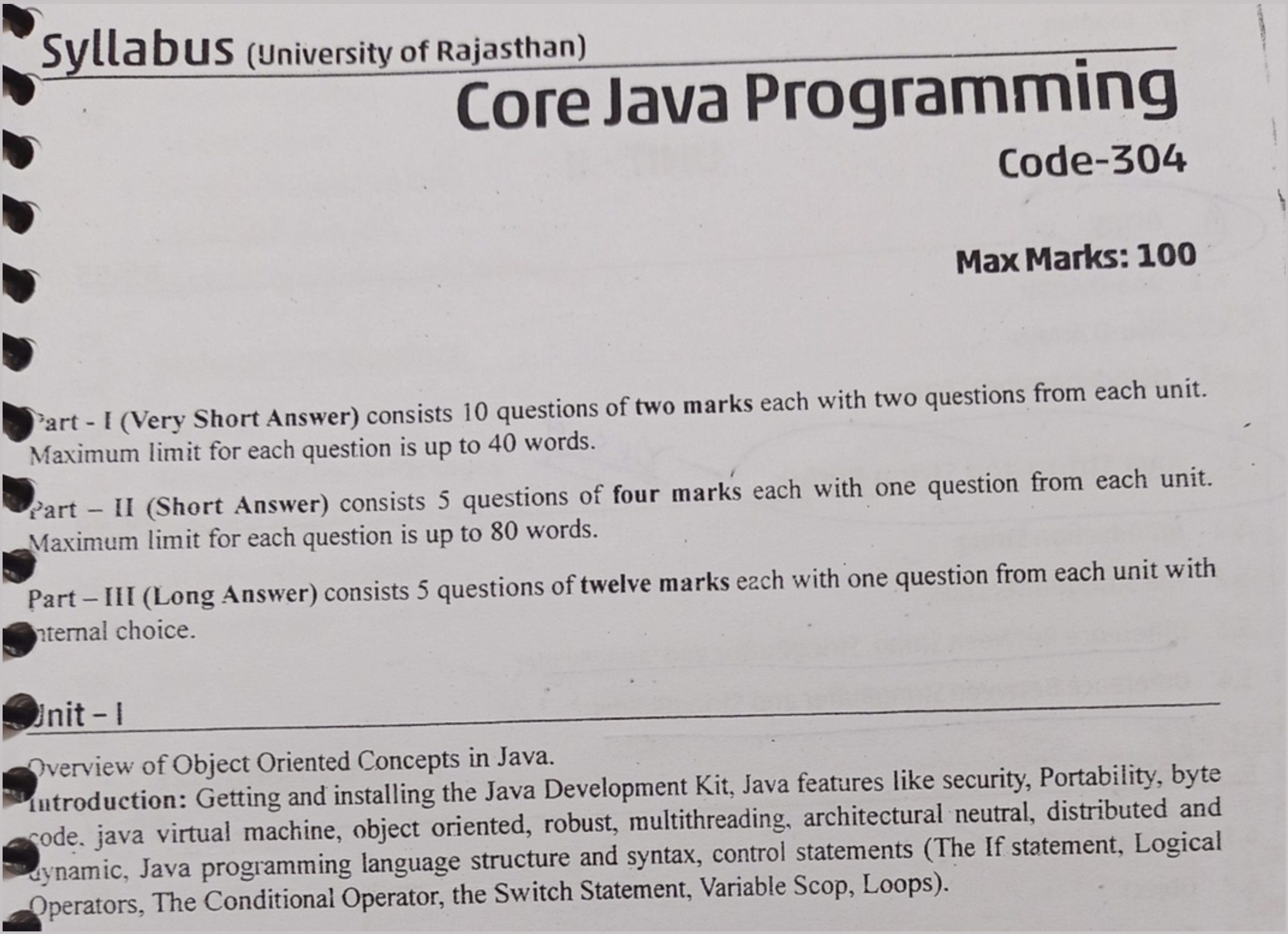
As per latest syllabus of University of Rajasthan, Jaipur; R.R.B.M. University, Alwar; D.U.S. University, Sikar and M.S.B. University, Bharatpur for BCA Part III

Jayanti Goyal

Head, Computer Department
Kanoria Girls College
Jaipur

Savita Singh

Assistant Professor, Computer Department
Kanoria Girls College
Jaipur



Syllabus (University of Rajasthan)

Core Java Programming

Code-304

Max Marks: 100

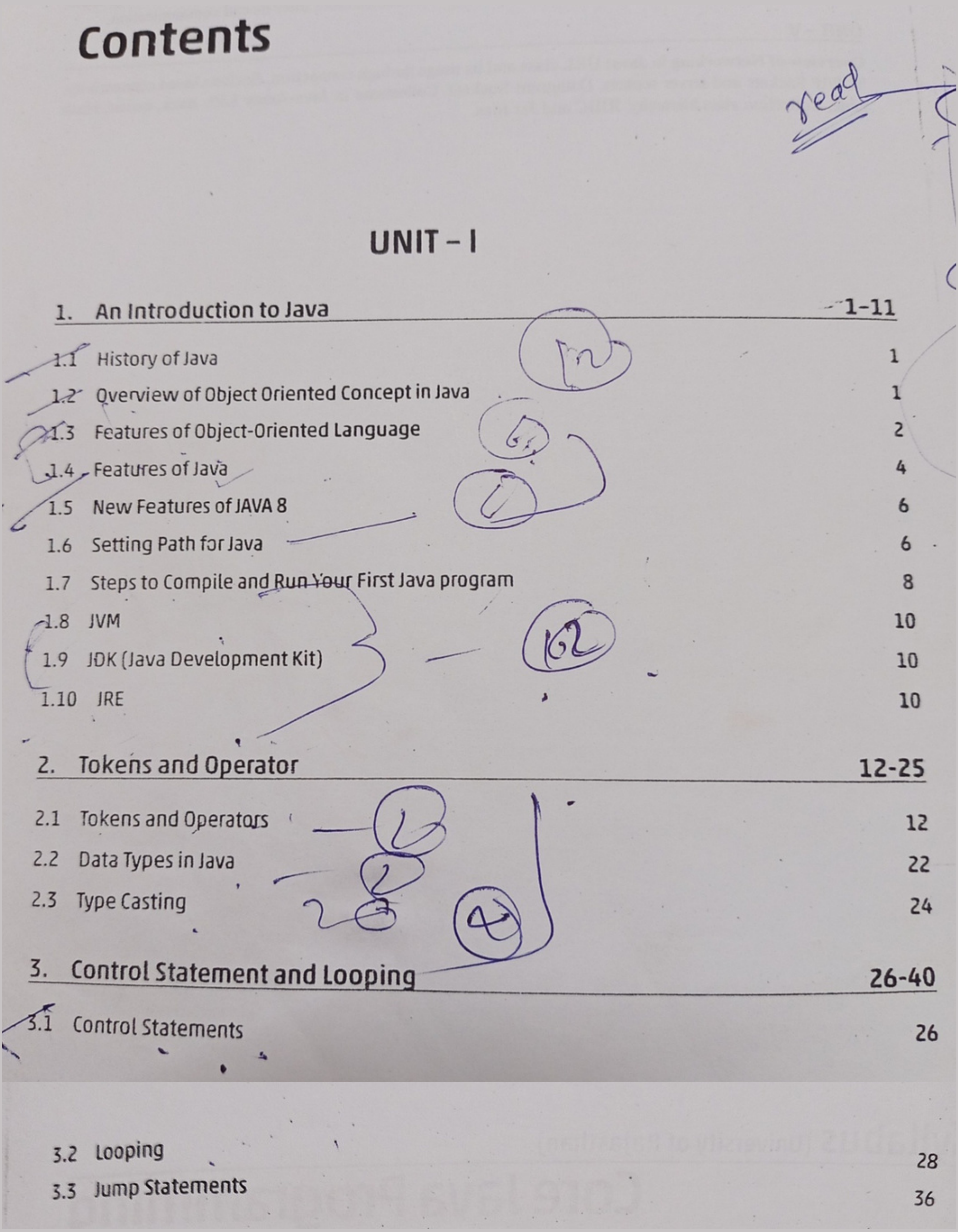
Part - I **(Very Short Answer)** consists 10 questions of **two marks** each with two questions from each unit. Maximum limit for each question is up to 40 words.

Part – II **(Short Answer)** consists 5 questions of **four marks** each with one question from each unit. Maximum limit for each question is up to 80 words.

Part – III **(Long Answer)** consists 5 questions of **twelve marks** each with one question from each unit with nternal choice.

Unit – I

Overview of Object Oriented Concepts in Java.
Introduction: Getting and installing the Java Development Kit, Java features like security, Portability, byte code, java virtual machine, object oriented, robust, multithreading, architectural neutral, distributed and dynamic, Java programming language structure and syntax, control statements (The If statement, Logical Operators, The Conditional Operator, the Switch Statement, Variable Scop, Loops).



Contents

read

UNIT – I

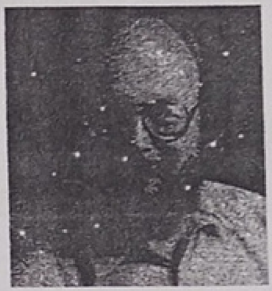
CHAPTER » 1

An Introduction to Java

1.1 History of Java

Java team members, also known as *Green Team*, initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, Java was suited for internet programming. Later, Java technology as incorporated by Netscape.



James Gosling

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. *James Gosling, Mike Sheridan, Ed Frank, Chrish Worth* and *Patrick Naughton* initiated the Java language project in June 1991. The small team of Sun Microsoft engineers called *Green Team*. Java, originally designed for small, embedded systems in electronic appliances like set-top boxes. Initially it was called *Oak* as Oak is a symbol of strength and choose as a national tree of many countries like U.S.A., France, Germany, Romania and was developed as a part of the Green project.

In 1995, *Oak* was renamed as *"Java"* because it was already a trademark by Oak Technologies. Java is an island of Indonesia where first coffee was produced (called java coffee).

Java, originally developed by James Gosling at Sun Microsystems, which is now a subsidiary of Oracle Corporation from 2010, and released in 1995. In 1995, Time magazine called *Java one of the Ten Best Products of 1995*. JDK 1.0 released in January 23, 1996.

1.2 Overview of Object Oriented Concept in Java

Object orientation ("OO"), refers to a method of programming and language design. The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. Data and code are combined into entities, called objects. An object can be thought of as contained bundle of behavior (code) and state (data).

A primary goal of "OO" programming is to develop more generic objects so that software can become more reusable between projects. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

Procedural programming languages are based on the paradigm of procedures. Object oriented programming models the real world in terms of objects. Everything in the world can be modeled as an object. A circle is an object, a person is an object, and a window icon is an object. A Java program is a object-oriented language, because programming in Java is centered on creating objects, manipulating objects, and making objects work together.

Advantage of OOPs over Procedure-oriented programming language

- OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

1.3 Features of Object-Oriented Language

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepis:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

1.3.1 Object

Any entity that has state and behavior is known as an object. An object in OOP has some state and behavior. In Java, the state is the set of values of an object's variables at any particular time and the behaviour of an object is implemented as methods.

1.3.2 Class

Collection of objects is called class. It is a logical entity. Class can be considered as the blueprint or a template for an object and describes the properties and behavior of that object, but without any actual existence. An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class.

1.3.3 Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism. Inheritance describes the parent child relationship between two classes.

A class can get some of its characteristics from a parent class and then add more unique features of its own. For example, consider a Vehicle is parent class and a child class's Car. Vehicle class will have properties and functionalities common for all vehicles. Car will inherit those common properties from the Vehicle class and then add properties which are specific to a Car.

In the above example, Vehicle parent class is known as base class or superclass. Car is known as derived class, Child class or subclass.

Java supports single-parent, multiple-children inheritance and multilevel inheritance (Grandparent-> Parent -> Child) for classes and interfaces.

Java supports multiple inheritance (multiple parents, single child) only through interfaces. This is done to avoid some confusions and errors such as diamond problem of inheritance.

1.3.4 Polymorphism

When one task is performed by different ways is known as polymorphism. In java, we use method overloading and method overriding to achieve polymorphism.

The ability to change form is known as polymorphism. Java supports different kinds of polymorphism like overloading and overriding.

1.3.4.1 Overloading

The same method name (method overloading) or operator symbol (operator overloading) can be used in different contexts. Java doesn't allow operator overloading except that "+" is overloaded for class String. The "+" operator can be used for addition as well as string concatenation. Overloading may be also called compile time polymorphism.

If a class has multiple methods by same name but different in number of argument, differ in type of argument and differ in order of argument then that is known as *Method Overloading*.

While calling an overloaded method it is possible that type of actual parameter passed may not be exactly with the formal parameter of any of the overloaded methods. In that cases parameters are promoted to next higher type till a match is found. If no match is found even after promoting the parameters then there is compile time error.

1.3.4.2 Overriding (or Subtype Polymorphism)

We can override an instance method of parent class in the child class. When you refer to a child class object using a Parent reference (e.g. Parent p = new Child()) and invoke a method, the overridden child class method will be invoked. Here, the actual method called will depend on the object at runtime, not the reference type.

Overriding is not applicable for static methods or variables (static and non-static). In case of variables (static and non-static) and static methods, when you invoke a method using a reference type variable, the method or variable that belong to the reference type is invoked.

When derived class have a method having same signature as that of base class method then, when we call the method by child class object then child class method is invoked not the base class, so the parent (base) class object is override by derived class. But when we call any method which is not define with base class method's signature then by child class object then base class method is invoked. Overriding may be also called runtime polymorphism.

1.3.5 Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

In plain English, abstract is a concept or idea not associated with any specific instance and does not have a concrete existence.

Abstraction in Object Oriented Programming refers to the ability to make a class abstract.

Abstraction captures only those details about an object that are relevant to the current, So that the programmer can focus on a few concepts at a time.

Java provides interfaces and abstract classes for describing abstract types.

- An *interface* is a contract or specification without any implementation. An interface can't have behavior or state.
- An *abstract class* is a class that cannot be instantiated, but has all the properties of a class including constructors. Abstract classes can have state and can be used to provide a skeletal implementation.

1.3.6 Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example, capsule is wrapped with different medicines. Encapsulation is the process of wrapping up of data (properties) and behavior (methods) of an object into a single unit; and the unit here is a Class (or interface).

Encaptulation, in plain English means *to enclose or be enclosed in or as if in a capsule*. In Java, everything is enclosed within a class or interface, unlike languages such as C and C++ where we can have global variables outside classes.

Encapsulation enables *data hiding*, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user. We can expose our operations hiding the details of what is needed to perform that operation.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

1.4 Features of Java

Java is a recently developed, concurrent, class-based, object-oriented programming and runtime environment, consisting of:

- A programming language.
- An API specification.
- A virtual machine specification.

1.4.1 Simple Language

Java is easy to learn and its syntax is quite simple, clean and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way. Eg : Pointers and Operator Overloading are not there in java but these are important part of C++.

1.4.2 Object Oriented

Java provides the basic object technology of C++ with some enhancements and some deletions. And Object oriented features are discussed earlier.

1.4.3 Architecture Neutral

Java source code is compiled into architecture-independent object code. The object code is interpreted by a Java Virtual Machine (JVM) on the target architecture.

1.4.4 Portable

Java Byte code can be carried to any platform. There is no implementation dependent features. Everything related to storage is predefined, example: size of primitive data types. Java implements additional portability standards. For example, `int`s are always 32-bit. User interfaces are built through an abstract window system that is readily implemented in Solaris and other operating environments.

1.4.5 Distributed and Dynamic

Java contains extensive TCP/IP networking facilities. Library routines support protocols such as HyperText Transfer Protocol (HTTP) and file transfer protocol (FTP). We can create distributed applications in java. RMI (Remote Method Invocation) and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

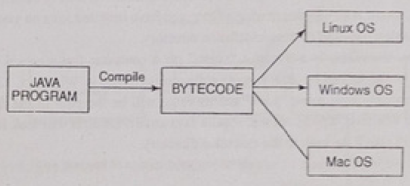
1.4.6 Robust

Both the Java compiler and the Java interpreter provide extensive error checking. Java manages all dynamic memory, checks array bounds, and other exceptions. Java uses strong memory management. There is lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust. Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking.

1.4.7 Platform Independent

Unlike other programming languages such as C, C++ etc. which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode (that is .class file). This bytecode is platform independent and can be run on any machine, in addition to this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



1.4.8 Secure

Features of C and C++ that often result in illegal memory accesses are not in the Java language. The interpreter also applies several tests to the compiled code to check for illegal code. After these tests, the compiled code causes no operand stack over- or underflows, performs no illegal data conversions, performs only legal object field accesses, and all these parameter types are verified as legal. Java is secured because:

- No explicit pointer.
- Programs run inside virtual machine sandbox.
- **Classloader** - It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier** - It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager** - It determines what resources a class can access such as reading and writing to the local disk.
- **High Performance** - Compilation of programs to an architecture independent machine-like language, results in a small efficient interpreter of Java programs. The Java environment also compiles the Java bytecode into native machine code at runtime.

1.4.9 Multithreaded

Multithreading can improve interactive performance by allowing operations, such as loading an image, to be performed while continuing to process user actions. A thread is like a separate program, which is executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

1.5 New Features of JAVA 8

Some of the core upgrades done as a part of Java 8 release are:

- Enhanced Productivity by providing Optional Classes feature, Lamda Expressions, Streams etc.
- Improved Polyglot programming. A *ploygot* is a program or script, written in a form which is valid in multiple programming languages and it performs the same operations in multiple programming languages. So Java now supports such type of programming technique.
- Improved Security and performance.

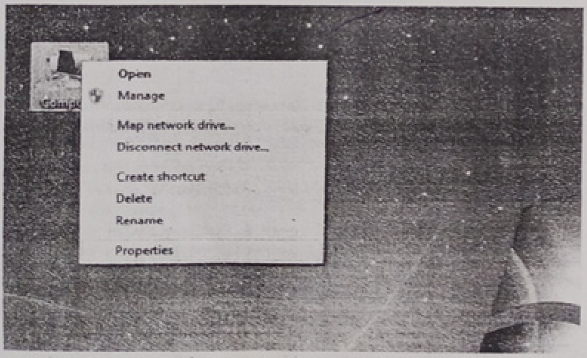
1.6 Setting Path for Java

Java is freely available on Oracle's Website. *Download the latest version of JDK* (Java Development Kit) on your machine. Install JDK on your machine. Once you have installed Java on your machine it need to set environment variable to point to correct installation directory.

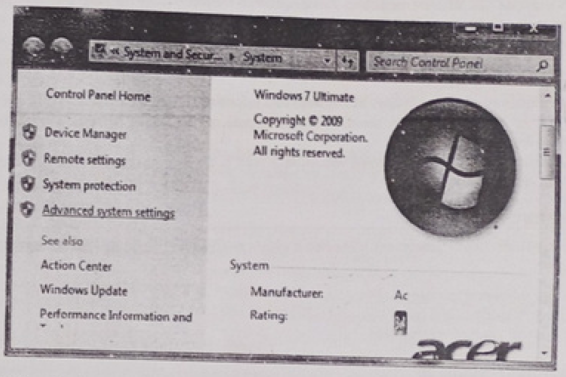
An *Environment variable* is a dynamic "object" on a computer that stores a value (like a key-value pair), which can be referenced by one or more software programs in Windows. Like in Java, we will set an environment variable with name "*java*" and its value will be the path of the */bin* directory present in Java directory. So whenever a program will require Java environment, it will look for the *java* environment variable which will give it the path to the execution directory.

After Java have installed in `C:\ Program files\Java \JDK directory` do following steps for path set.

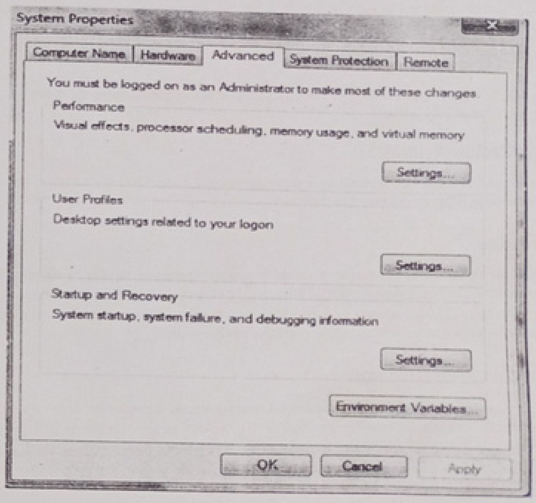
Step 1: Right click on my computer and select properties.



Step 2: Go to the Advance System Settings tab.



Step 3: Click on Environment Variables button.


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

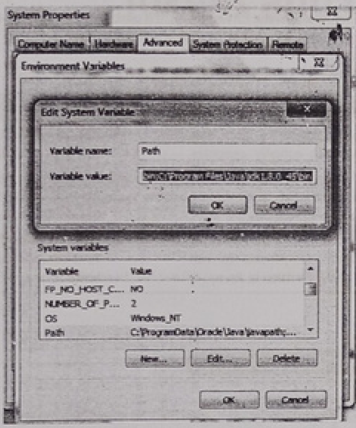
User Profiles
Desktop settings related to your logon
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK | Cancel | Apply

Step 4: Now alter the path variable so that it also contains the path to JDK installed directory.



Variable name: Path

Variable value: bin;C:\Program Files\Java\jdk1.8.0_45\bin

Variable	Value
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;...

First Java Program

Let us look at a simple java program.

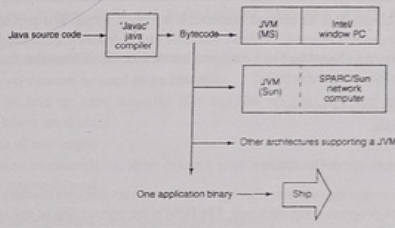
```
class Hello
{
  public static void main(String[] args)
  {
    System.out.println ("Hello Java program");
  }
}
```

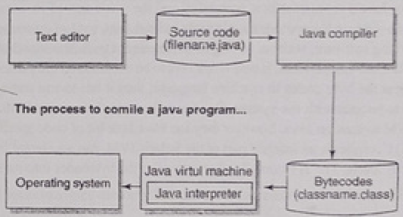
class: class keyword is used to declare classes in Java
public: It is an access specifier. Public means this function is visible to all.
static: static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The *main()* method here is called by JVM, without creating any object for class.
void: It is the return type, meaning this function will not return anything.
main(): main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.
System out println: This is used to print anything on the console like printf() in C language.

1.7 Steps to Compile and Run Your First Java Program

Step 1: Open a text editor and write the code as above.
Step 2: Save the file as Hello.java

Step 3: Open command prompt and go to the directory where you saved your first java program assuming it is saved in `C:\`
Step 4: Type *javac Hello.java* and press Return to compile your code. This command will call the Java Compiler asking it to compile the specified file. If there are no errors in the code the command prompt will take you to the next line.
Step 5: Now type *java Hello* on command prompt to run your program.
Step 6: You will be able to see *Hello java program* printed on your command prompt.

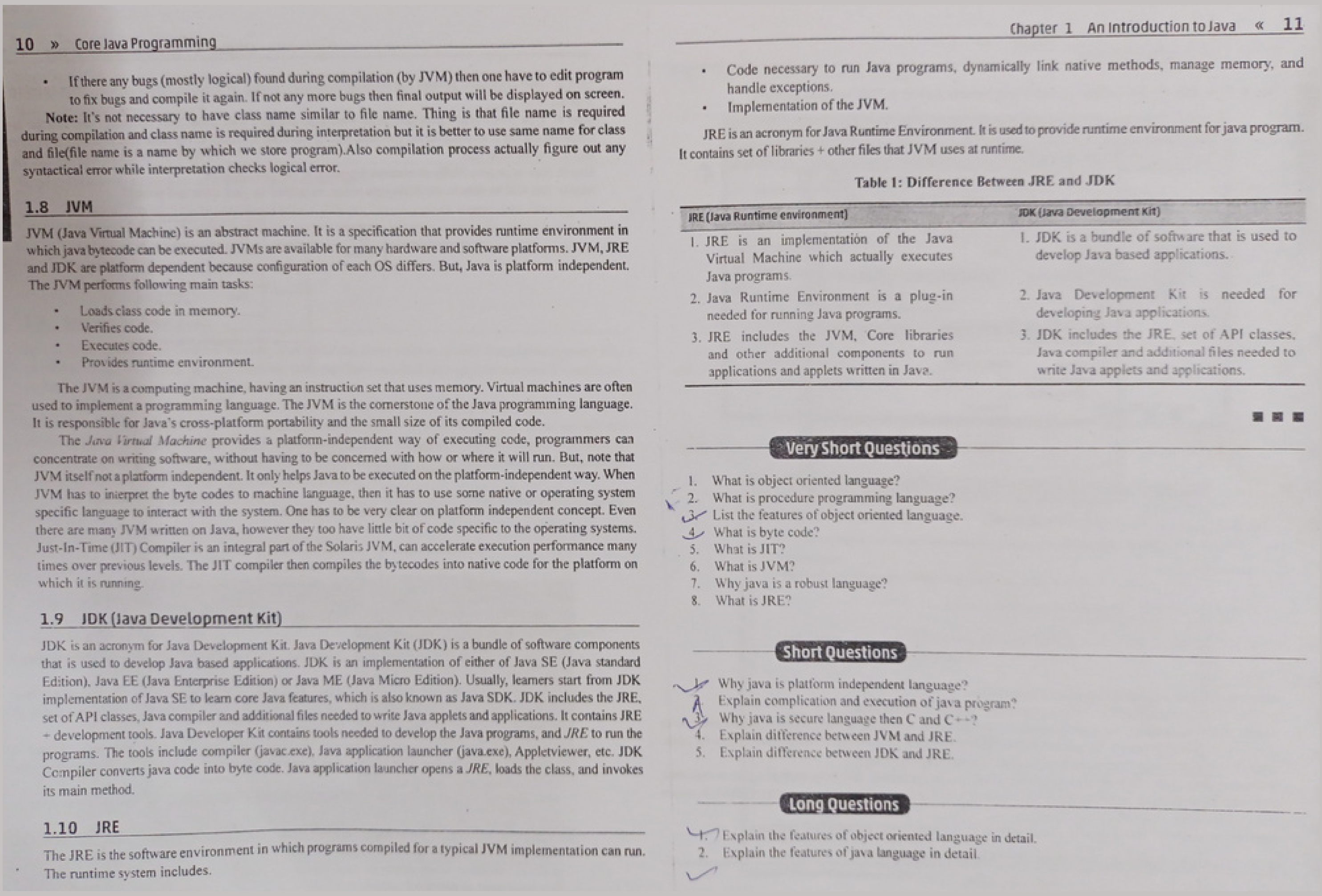




The process to comile a java program...

Compilation and Execution Process:

- Create a program using editor (eg. Notepad) and save it as filename with .java extension specified directory.
- Then we compile it using javac command (eg. javac filename.java), A java compiler then convert .java file into .class file which is called as bytecode and this bytecode is platform independent can run on different platforms eg.Windows, Linux,MAC etc.
- If any bugs found during compilation that should be fixed first then compiled again. If there is not any bugs found then bytecode is created and handed over to JVM (java virtual machine) for interpretation.
- JVM interpret class file by running java space class name command (eg. java filename) as given in figure.
- JVM has three components to interpret program first: Bytecode verifier which verifies bytecode, second: Class loader which loads required classes from java class library, third one is a JIT (just in time) translator which compiles class file and convert it into machine readable format and then hand over to operating system.



- If there any bugs (mostly logical) found during compilation (by JVM) then one have to edit program to fix bugs and compile it again. If not any more bugs then final output will be displayed on screen.

Note: It's not necessary to have class name similar to file name. Thing is that file name is required during compilation and class name is required during interpretation but it is better to use same name for class and file(file name is a name by which we store program).Also compilation process actually figure out any syntactical error while interpretation checks logical error.

1.8 JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent. The JVM performs following main tasks:

- Loads class code in memory.
- Verifies code.
- Executes code.
- Provides runtime environment.

The JVM is a computing machine, having an instruction set that uses memory. Virtual machines are often used to implement a programming language. The JVM is the cornerstone of the Java programming language. It is responsible for Java's cross-platform portability and the small size of its compiled code.

The *Java Virtual Machine* provides a platform-independent way of executing code, programmers can concentrate on writing software, without having to be concerned with how or where it will run. But, note that JVM itself not a platform independent. It only helps Java to be executed on the platform-independent way. When JVM has to interpret the byte codes to machine language, then it has to use some native or operating system specific language to interact with the system. One has to be very clear on platform independent concept. Even there are many JVM written on Java, however they too have little bit of code specific to the operating systems. Just-In-Time (JIT) Compiler is an integral part of the Solaris JVM, can accelerate execution performance many times over previous levels. The JIT compiler then compiles the bytecodes into native code for the platform on which it is running.

1.9 JDK (Java Development Kit)

JDK is an acronym for Java Development Kit. Java Development Kit (JDK) is a bundle of software components that is used to develop Java based applications. JDK is an implementation of either of Java SE (Java standard Edition), Java EE (Java Enterprise Edition) or Java ME (Java Micro Edition). Usually, learners start from JDK implementation of Java SE to learn core Java features, which is also known as Java SDK. JDK includes the JRE, set of API classes, Java compiler and additional files needed to write Java applets and applications. It contains JRE + development tools. Java Developer Kit contains tools needed to develop the Java programs, and *JRE* to run the programs. The tools include compiler (javac.exe), Java application launcher (java.exe), Appletviewer, etc. JDK Compiler converts java code into byte code. Java application launcher opens a *JRE*, loads the class, and invokes its main method.

1.10 JRE

The JRE is the software environment in which programs compiled for a typical JVM implementation can run. The runtime system includes.

- Code necessary to run Java programs, dynamically link native methods, manage memory, and handle exceptions.
- Implementation of the JVM.

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment for java program. It contains set of libraries + other files that JVM uses at runtime.

Table 1: Difference Between JRE and JDK

JRE (Java Runtime environment)	JDK (Java Development Kit)
1. JRE is an implementation of the Java Virtual Machine which actually executes Java programs.	1. JDK is a bundle of software that is used to develop Java based applications.
2. Java Runtime Environment is a plug-in needed for running Java programs.	2. Java Development Kit is needed for developing Java applications.
3. JRE includes the JVM, Core libraries and other additional components to run applications and applets written in Java.	3. JDK includes the JRE, set of API classes, Java compiler and additional files needed to write Java applets and applications.

Very Short Questions

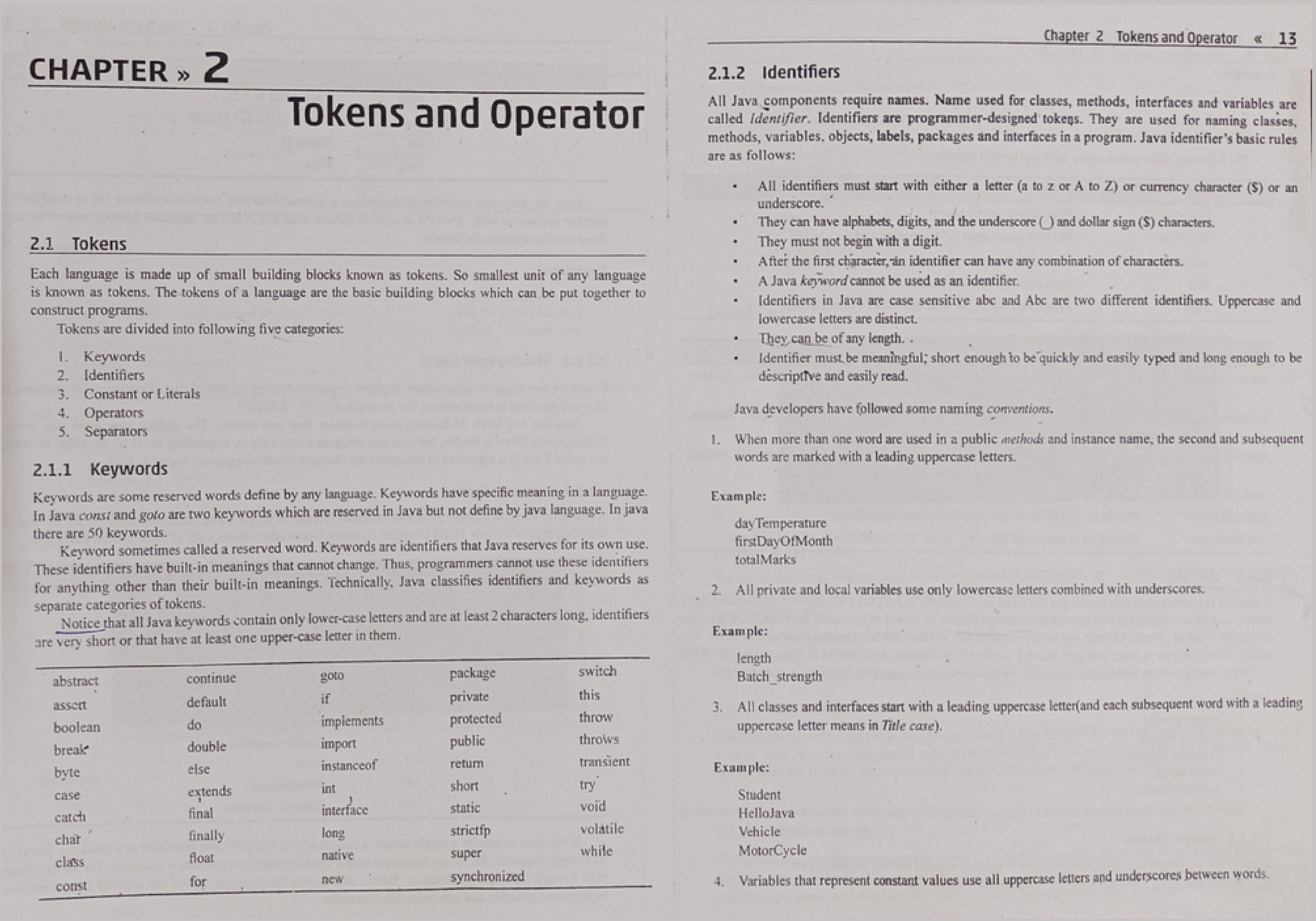
1. What is object oriented language?
2. What is procedure programming language?
3. List the features of object oriented language.
4. What is byte code?
5. What is JIT?
6. What is JVM?
7. Why java is a robust language?
8. What is JRE?

Short Questions

1. Why java is platform independent language?
2. Explain complication and execution of java program?
3. Why java is secure language then C and C++?
4. Explain difference between JVM and JRE.
5. Explain difference between JDK and JRE.

Long Questions

1. Explain the features of object oriented language in detail.
2. Explain the features of java language in detail.



CHAPTER » 2

Tokens and Operator

2.1 Tokens

Each language is made up of small building blocks known as tokens. So smallest unit of any language is known as tokens. The tokens of a language are the basic building blocks which can be put together to construct programs.

Tokens are divided into following five categories:

1. Keywords
2. Identifiers
3. Constant or Literals
4. Operators
5. Separators

2.1.1 Keywords

Keywords are some reserved words define by any language. Keywords have specific meaning in a language. In Java *const* and *goto* are two keywords which are reserved in Java but not define by java language. In java there are 50 keywords.

Keyword sometimes called a reserved word. Keywords are identifiers that Java reserves for its own use. These identifiers have built-in meanings that cannot change. Thus, programmers cannot use these identifiers for anything other than their built-in meanings. Technically, Java classifies identifiers and keywords as separate categories of tokens.

Notice that all Java keywords contain only lower-case letters and are at least 2 characters long, identifiers are very short or that have at least one upper-case letter in them.

abstract	continue	goto	package	switch
assert	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	synchronized	

2.1.2 Identifiers

All Java components require names. Name used for classes, methods, interfaces and variables are called *Identifier*. Identifiers are programmer-designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifier's basic rules are as follows:

- All identifiers must start with either a letter (a to z or A to Z) or currency character ($) or an underscore.
- They can have alphabets, digits, and the underscore (_) and dollar sign ($) characters.
- They must not begin with a digit.
- After the first character, an identifier can have any combination of characters.
- A Java *keyword* cannot be used as an identifier.
- Identifiers in Java are case sensitive abc and Abc are two different identifiers. Uppercase and lowercase letters are distinct.
- They can be of any length. .
- Identifier must be meaningful; short enough to be quickly and easily typed and long enough to be descriptive and easily read.

Java developers have followed some naming *conventions*.

1. When more than one word are used in a public *methods* and instance name, the second and subsequent words are marked with a leading uppercase letters.

Example:

dayTemperature
firstDayOfMonth
totalMarks

2. All private and local variables use only lowercase letters combined with underscores.

Example:

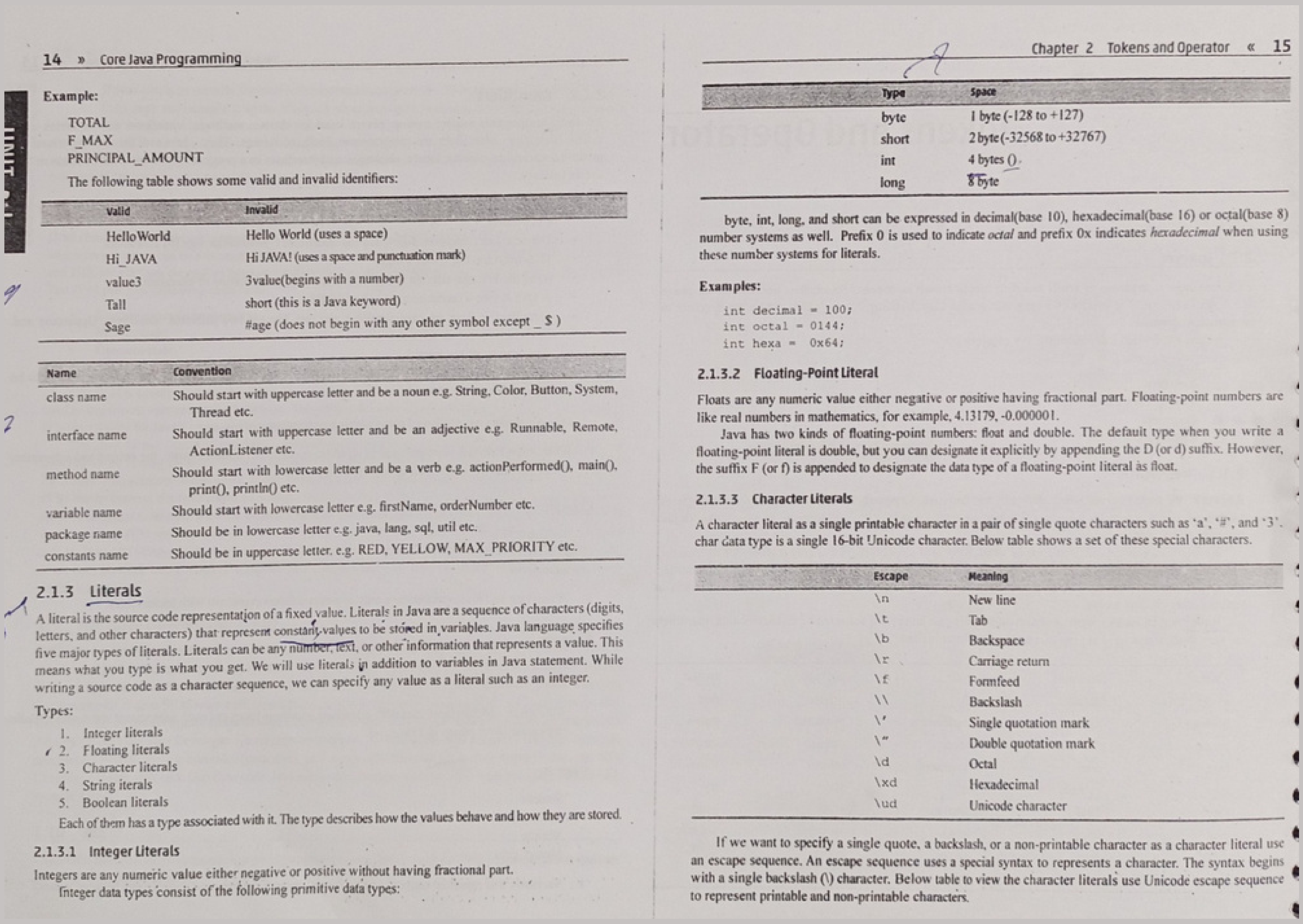
length
Batch_strength

3. All classes and interfaces start with a leading uppercase letter(and each subsequent word with a leading uppercase letter means in *Title case*).

Example:

Student
HelloJava
Vehicle
MotorCycle

4. Variables that represent constant values use all uppercase letters and underscores between words.



Example:

TOTAL
F_MAX
PRINCIPAL_AMOUNT

The following table shows some valid and invalid identifiers:

Valid	Invalid
HelloWorld	Hello World (uses a space)
Hi_JAVA	Hi JAVA! (uses a space and punctuation mark)
value3	3value(begins with a number)
Tall	short (this is a Java keyword)
Sage	#age (does not begin with any other symbol except _ $)

Name	Convention
class name	Should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	Should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	Should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	Should start with lowercase letter e.g. firstName, orderNumber etc.
package name	Should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	Should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

2.1.3 Literals

A literal is the source code representation of a fixed value. Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java language specifies five major types of literals. Literals can be any number, text, or other information that represents a value. This means what you type is what you get. We will use literals in addition to variables in Java statement. While writing a source code as a character sequence, we can specify any value as a literal such as an integer.

Types:

1. Integer literals
2. Floating literals
3. Character literals
4. String iterals
5. Boolean literals

Each of them has a type associated with it. The type describes how the values behave and how they are stored.

2.1.3.1 Integer Literals

Integers are any numeric value either negative or positive without having fractional part.

Integer data types consist of the following primitive data types:

Type	Space
byte	1 byte (-128 to +127)
short	2 byte(-32568 to +32767)
int	4 bytes ().
long	8 byte

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well. Prefix 0 is used to indicate *octal* and prefix 0x indicates *hexadecimal* when using these number systems for literals.

Examples:

```
int decimal = 100;
int octal = 0144;
int hexa =  0x64;
```

2.1.3.2 Floating-Point Literal

Floats are any numeric value either negative or positive having fractional part. Floating-point numbers are like real numbers in mathematics, for example, 4.13179, -0.000001.

Java has two kinds of floating-point numbers: float and double. The default type when you write a floating-point literal is double, but you can designate it explicitly by appending the D (or d) suffix. However, the suffix F (or f) is appended to designate the data type of a floating-point literal as float.

2.1.3.3 Character Literals

A character literal as a single printable character in a pair of single quote characters such as 'a', '#', and '3'. char data type is a single 16-bit Unicode character. Below table shows a set of these special characters.

Escape	Meaning
\n	New line
\t	Tab
\b	Backspace
\r	Carriage return
\f	Formfeed
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\d	Octal
\xd	Hexadecimal
\ud	Unicode character

If we want to specify a single quote, a backslash, or a non-printable character as a character literal use an escape sequence. An escape sequence uses a special syntax to represents a character. The syntax begins with a single backslash (\) character. Below table to view the character literals use Unicode escape sequence to represent printable and non-printable characters.

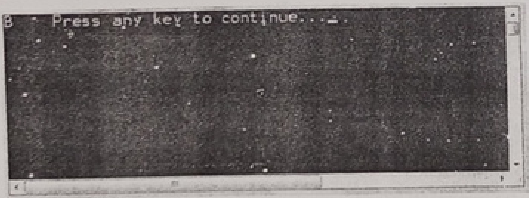
'u0041'	Capital letter A
'\u0030'	Digit 0
'\u0022'	Double quote "
'\u003b'	Punctuation ;
'\u0020'	Space
'\u0009'	Horizontal Tab

Example 1:

```
class Char_1

{
  public static void main(String arg[])
  {
    char ch='b';
    if(ch>=97&&ch<=122)
    {
      ch=(ch-32);//ch=(ch-32);possible loss of precision  required: char found:
int
      //ch=97-32;
    }
   // ch=Character.toUpperCase(ch);
    System.out.print(ch+"    ");
  }
 }
```

Output :



```
B  Press any key to continue...
```

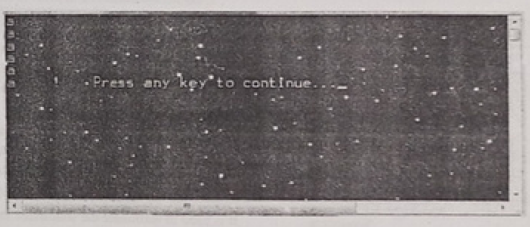
Example 2:

```
class Char_2

{
  public static void main(String arg[])
  {
    char ch='a';
    System.out.println(ch+"    ");
    ch=97;
```

```
    System.out.println(ch+"    ");
    ch=0141;//octal
    System.out.println(ch+"    ");
    ch=0x61;
    System.out.println(ch+"    ");
    ch=0b1100001;
    System.out.println(ch+"    ");
    ch='\u0061';
    System.out.print(ch+"    ");
    ch='\41';
    System.out.print(ch+"    ");
  }
}
```

Output:



```
a
a
a
a
a
a  !  Press any key to continue...
```

2.3.1.4 String Literals

The set of characters in represented as String literals in Java. Always use "double quotes" for String literals. There are few methods provided in Java to combine strings, modify strings and to know whether to strings have the same values.

"": The empty string
"\": A string containing one character
"This is a string" A string containing 16 characters

2.1.3.5 Null Literals

The final literal that we can use in Java programming is a Null literal. We specify the Null literal in the source code as 'null'. To reduce the number of references to an object, use null literal. The type of the null literal is always null. We typically assign null literals to object reference variables. For instance

```
s = null;
```

2.1.3.6 Boolean Literals

The values true and false are treated as literals in Java programming. When we assign a value to a boolean variable, we can only use these two values. Unlike C, we can't assume that the value of 1 is equivalent to true and 0 is equivalent to false in Java. We have to use the values true and false to represent a Boolean value.

Example:

```
boolean chosen = true;
boolean chosen = false;
```

Remember that the literal true is not represented by the quotation marks around it. The Java compiler will take it as a string of characters, if its in quotation marks.

2.1.4 Separators in Java

Separators help define the structure of a program. The separators used are parentheses ( ), braces { }, the period ., and the semicolon ;. The table lists the six Java separators (nine if you count opening and closing separators as two). Following are the some characters which are generally used as the separators in Java.

Separator	Name	Use
.	Period	It is used to separate the package name from sub-package name & class name. It is also used to separate variable or method from its object or instance.
,	Comma	It is used to separate the consecutive parameters in the method definition. It is also used to separate the consecutive variables of same type while declaration.
;	Semicolon	It is used to terminate the statement in Java.
()	Parenthesis	This holds the list of parameters in method definition. Also used in control statements & type casting
{}	Braces	This is used to define the block/scope of code, class, methods.
[]	Brackets	It is used in array declaration.

2.1.5 Comments

Comments are special part of any language, to describe or explain the code which make easy to understand. Comments are not increase the line of code as comments are not executed by the compilers. Comments are read by compilers but not compile by any compiler. So the byte code does not contain the respective code for comments.

Type of Comments in Java:

1. Single Line Comments
2. Multiple Line Comments
3. Documents Comments

2.1.5.1 Single Line Comments

Java's single line comment starts with two forward slashes with no white spaces (//) and lasts till the end of line. If the comment exceeds one line then put two more consecutive slashes on next line and continue the comment.

Java's single line comments are proved useful for supplying short explanations for variables, function declarations, and expressions.

Example:

```
if(x < y)
{ // begin if block
  x = y;
  y = 0;
} // end if block
```

2.1.5.2 Java Multi-Line Comments or Slash-star Comments

Java's multi-line or slash-star or traditional comment is a piece of text enclosed in slash-star (/*) and star-slash (*/). Again there should be no white space between slash and star. Java's multi-line comments are useful when the comment text does not fit into one line; therefore needs to span across lines.

Example 3:

```
public class Abc
{
  public static void main(String[] args)
  {
    for (int i = 0; i < 5; /* exits when i reaches to 5 */ i++)
    {
      System.out.print(i + " ");
    }
  }
}
```

Output

```
======
0 1 2 3 4
```

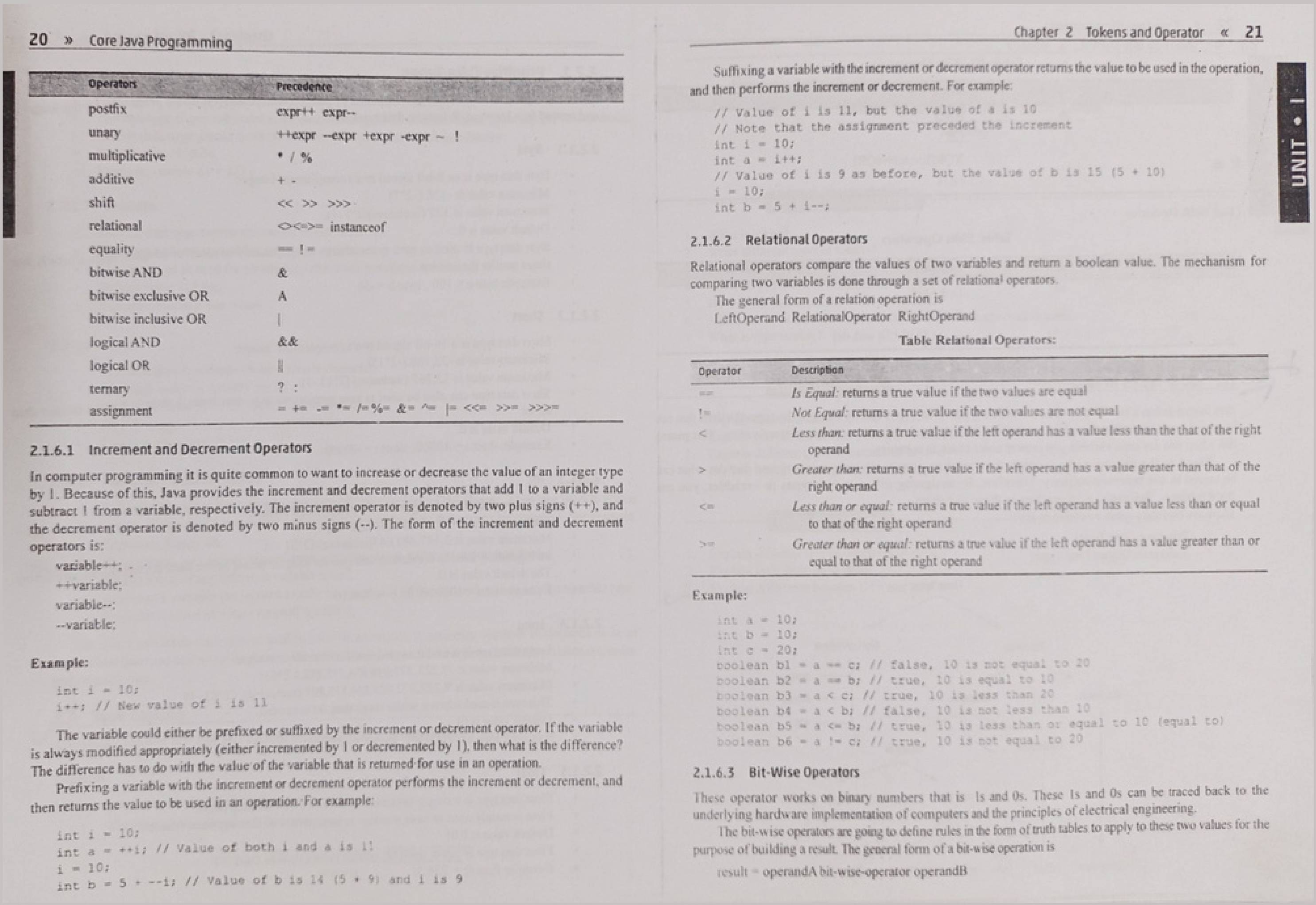
2.1.5.3 Document Comment

```
/** documentation */
```

This is a documentation comment and in general its called *doc comment*. The *JDK javadoc* tool uses *doc comments* when preparing automatically generated documentation.

2.1.6 Operators

Operator in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.



Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	<><=>= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	A
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /=%= &= ^= |= <<= >>= >>>=

2.1.6.1 Increment and Decrement Operators

In computer programming it is quite common to want to increase or decrease the value of an integer type by 1. Because of this, Java provides the increment and decrement operators that add 1 to a variable and subtract 1 from a variable, respectively. The increment operator is denoted by two plus signs (++), and the decrement operator is denoted by two minus signs (--). The form of the increment and decrement operators is:

```
variable++;
++variable;
variable--;
--variable;
```

Example:

```
int i = 10;
i++; // New value of i is 11
```

The variable could either be prefixed or suffixed by the increment or decrement operator. If the variable is always modified appropriately (either incremented by 1 or decremented by 1), then what is the difference? The difference has to do with the value of the variable that is returned for use in an operation.

Prefixing a variable with the increment or decrement operator performs the increment or decrement, and then returns the value to be used in an operation. For example:

```
int i = 10;
int a = ++i; // Value of both i and a is 11
i = 10;
int b = 5 + --i; // Value of b is 14 (5 + 9) and i is 9
```

Suffixing a variable with the increment or decrement operator returns the value to be used in the operation, and then performs the increment or decrement. For example:

```
// Value of i is 11, but the value of a is 10
// Note that the assignment preceded the increment
int i = 10;
int a = i++;
// Value of i is 9 as before, but the value of b is 15 (5 + 10)
i = 10;
int b = 5 + i--;
```

2.1.6.2 Relational Operators

Relational operators compare the values of two variables and return a boolean value. The mechanism for comparing two variables is done through a set of relational operators.

The general form of a relation operation is

LeftOperand RelationalOperator RightOperand

Table Relational Operators:

Operator	Description
==	*Is Equal:* returns a true value if the two values are equal
!=	*Not Equal:* returns a true value if the two values are not equal
<	*Less than:* returns a true value if the left operand has a value less than the that of the right operand
>	*Greater than:* returns a true value if the left operand has a value greater than that of the right operand
<=	*Less than or equal:* returns a true value if the left operand has a value less than or equal to that of the right operand
>=	*Greater than or equal:* returns a true value if the left operand has a value greater than or equal to that of the right operand

Example:

```
int a = 10;
int b = 10;
int c = 20;
boolean b1 = a == c; // false, 10 is not equal to 20
boolean b2 = a == b; // true, 10 is equal to 10
boolean b3 = a < c; // true, 10 is less than 20
boolean b4 = a < b; // false, 10 is not less than 10
boolean b5 = a <= b; // true, 10 is less than or equal to 10 (equal to)
boolean b6 = a != c; // true, 10 is not equal to 20
```

2.1.6.3 Bit-Wise Operators

These operator works on binary numbers that is 1s and 0s. These 1s and 0s can be traced back to the underlying hardware implementation of computers and the principles of electrical engineering.

The bit-wise operators are going to define rules in the form of truth tables to apply to these two values for the purpose of building a result. The general form of a bit-wise operation is

result = operandA bit-wise-operator operandB

Table: Bit-Wise Operators

Operator	Description
&	AND
|	OR
^	XOR(ExclusiveOR)
~	NOT

Left Shift Operator

Table: Shift Operators

Operator	Description
<<	Left Shift
>>	Right Shift
>>>	Right Shift (fill with 0s)

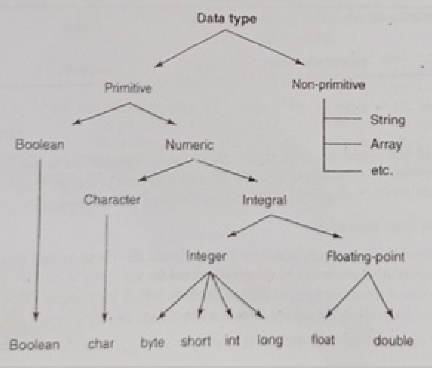
2.2 Data Types in Java

Java language has a rich implementation of data types. Data types specify size and the type of values that can be stored in an identifier. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types



2.2.1 Primitive Data Types

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Primitive data types are:

2.2.1.1 Byte

- Byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7).
- Maximum value is 127 (inclusive)(2^7 -1).
- Default value is 0.
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example: byte a = 100 , byte b = -50

2.2.1.2 Short

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15).
- Maximum value is 32,767 (inclusive) (2^15 -1).
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example: short s = 10000, short r = -20000

2.2.1.3 Int

- int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000

2.2.1.4 Long

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, long b = -200000L

2.2.1.5 Float

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example: float f1 = 234.5f

2.2.1.6 Double

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

2.2.1.7 Boolean

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example: boolean one = true

2.2.1.8 Char

- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA ='A'

2.2.2 Reference Data Types

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: Animal a1 = new Animal("giraffe");

So, a1is a reference data type is used to refer to an object. A reference variable is declared to be of specific and that type can never be change. We will talk a lot more about reference data type later in Classes and Object lesson.

✓ 2.3 Type Casting

Assigning a value of one type to a variable of another type is known as *Type Casting.*

Example:

In Java, type casting is classified into two types.

- Widening Casting(Implicit)/ Upcasting

$$\begin{tikzcd} \text{byte} \arrow[r] & \text{Short} \arrow[r] & \text{int} \arrow[r] & \text{long} \arrow[r] & \text{float} \arrow[r] & \text{double} \end{tikzcd}$$

$$\xrightarrow[\text{widening}]{}$$

- Narrowing Casting(Explicitly done)/ down casting

$$\begin{tikzcd} \text{double} \arrow[r] & \text{float} \arrow[r] & \text{long} \arrow[r] & \text{int} \arrow[r] & \text{short} \arrow[r] & \text{byte} \end{tikzcd}$$

$$\xrightarrow[\text{Narrowing}]{}$$

■ ■ ■

Very Short Questions

1. What is token?
2. Write different types of tokens?
3. What is keyword?
4. What do you mean by comments?
5. How many data types are available in java?
6. Name the keyword which are not define in java but reserved in java.
7. What is type casting?

Short Questions

1. Explain tokens in detail.
2. Explain convention and rules for java identifiers.
3. Explain different types of comments in java.
4. Explain implicit and explicit type casting in java.

Long Questions

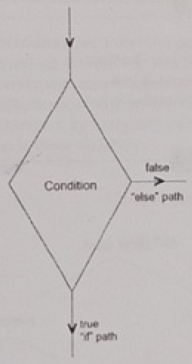
1. Explain different type of operators in java.
2. Explain java datatype in detail.
3. ✓ Explain difference between C++ and java.

CHAPTER » 3

Control Statement and Looping

3.1. Control Statements

A conditional statement allows a choice from a selection of statements. It first evaluates an expression to decide between the possibilities. If there are only two then a boolean valued expression (also called a logical expression) will be enough to allow us to choose. We can picture a conditional statement graphically. We have a condition box with two exit paths, one for true and the other for false. The true path leads to the "if"-statement. The false path leads to the "else"-statement. After these two statements the paths meet up so that control can pass to the next statement in the program.



3.1.1 If Statement

The *if* statement is the fundamental control statement that allows Java to make decisions and execute statements conditionally.

Syntax:

```
if (expression)
{
Statement or statements to be executed if expression is true
}
```

3.1.2 If...else Statement

The *if...else* statement is the next form of control statement that allows Java to execute statements in more controlled way.

Syntax:

```
if (expression)
{
Statement or statements to be executed if expression is true
else
{
Statement or statements to be executed if expression is false
}
```

3.1.3 if...else if... Statement

The *if...else if...* statement is the one level advance form of control statement that allows Java to make correct decision out of several conditions.

Syntax:

```
if (expression 1)
{
Statement(s) to be executed if expression 1 is true
}
else if (expression 2)
{
Statement(s) to be executed if expression 2 is true
}
else if (expression 3)
{
Statement(s) to be executed if expression 3 is true
}
else
{
Statement(s) to be executed if no expression is true
}
```

3.1.4 Switch Statement

The basic syntax of the *switch* statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each *case* against the value of the expression until a match is found. If nothing matches, then control goes to *default statement*.

Usually default is last statement, so 'break' is optional after default statement. 'default' can be use in between then we have use break.

Conclusion is that break is optional after last statement.

```
switch (expression)
{
case condition 1: statement(s)
break;
case condition 2: statement(s)
break;
...
case condition n: statement(s)
break;
default: statement(s)
}
```

3.2 Looping

Java performs several types of repetitive operations, called "looping". Loops are set of instructions used to repeat the same block of code till a specified condition returns false or true depending on requirement. To control the loops use counter variable that increments or decrements witn each repetition of the loop.

Java supports three loop statements: for, do-while and while. The for loops are best used when you want to perform a loop a specific number of times The while loop are best used to perform a loop an undetermined number of times. In addition, you can use the break and continue statements within loop statements.

If we want to execute a statement or group of statements are execute multiple times till any condition is reached, then we use loop.

Types of loops:

On the bases of number of iteration (that is repetitions of the body of the loop) loops are divided into two categories that are: *Definite* and *indefinite loops*

In definite loops, the number of iterations is known before we start the execution of the body of the loop.

Example: repeat for 10 times printing out a *.

In indefinite loops, the number of iterations is not known before we start to execute the body of the loop, but depends on when a certain condition becomes true (and this depends on what happens in the body of the loop)

Example. while the user does not decide it is time to stop, print out a * and ask the user whether he wants to stop.

3.2.1 While-Loop

The while loop is commonly used loop after the for loop. The while statement repeats a loop as long as a specified condition evaluates to true. If the condition becomes false, the statements within the loop stop executing and control passes to the statement following the loop. It is entry control loop, means first check condition then enter the loop. The while statement looks as follows:

Syntax:

```
while(condtion)
{
Statements.
...
...
}
```

- Condition is an expression of type boolean.
- Statements are a loop statement (also called the body of the loop)

Note: By making use of a block, it is possible to group several statements into a single composite statement, it is in fact possible to have more than one statement in the body of the loop.

Semantics:

- First, the condition is evaluated.
- If it is true, the statement is executed and the value of the condition is evaluated again, continuing in this way until the condition becomes false.
- At this point the statement immediately following the while loop is executed.

Hence, the body of the loop is executed as long as the condition stays true. As soon as it becomes false control comes out of the loop and continue with the following statement.

Example 1: Print out 100 stars.

```
int i = 0;
while (i < 100)
{
  System.out.print("*");
  i++;
}
```

Example 2: Print the squares of the integers between 1 and 10.

```
int i = 1;
while (i <= 10)
{
  System.out.println(i * i);
  i++;
}
```

The following are the common features of loops controlled by a counter:

- A control variable for the loop is used (also called counter or index of the loop). E.g. `int i;`
- Initialization of the control variable E.g., `int i = 1;`
- Increment (or decrement) of the control variable at each iteration E.g., `i++;`
- Test if we have reached the final value of the control variable E.g., `(i <= 10)`

3.2.2 do...while Loop

The *do...while* loop is similar to the *while* loop except that the condition check takes place at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false. It is exit control loop as first enter the loop and then check condition.

Syntax:

```
do
{
  Statements;
  .............
  .............
} while(condition);
```

Example 3: Sum integers read from input until a 0 is read.

```
int i;
int sum = 0;
```

```
do
{
  i = Integer.parseInt(JOptionPane.showInputDialog(
    "Input an integer (0 to terminate)"));
    sum = sum + i;
} while (i != 0);
System.out.println("sum = " + sum);
```

3.2.3 For Loop V.V.I

It is also an entry control loop, means first check condition then enter the loop. The for loop is executed till a specified condition returns false. It has basically the same syntax then in other languages. It takes 3 arguments. When the for loop executes, the following occurs:

1. The initializing expression is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the for loop terminates.
3. The update expression increment executes.
4. The statements execute, and control returns to step 2.

Syntax:

```
For(initialization; condition; increament/decreament/update)
{
Statement(s) to be executed if test condition is true
}
```

- ✓ Initialization is an expression with side-effect that initializes a control variable (typically an assignment), which can also be a declaration with initialization
- ✓ Condition is an expression of type boolean.
- ✓ Update is an expression with side-effect that typically consists in updating (i.e., incrementing or decrementing) the control variable.
- ✓ Statement is a single statement (also called body of the for loop)

```
{
  initialization ;
  while (condition )
  {
    Statement;
    update ;
  }
}
```

The for loop examples.

- `for (int i = 1; i <= 10; i++) ...`
  values assigned to i: 1, 2, 3, ..., 10
- `for (int i = 10; i >= 1; i--) ...`
  values assigned to i: 10, 9, 8, ..., 2, 1
- `for (int i = -4; i <= 4; i = i+2) ...`
  values assigned to i: -4, -2, 0, 2, 4
- for (int i = 0; i >= -10; i = i-3) ...
  values assigned to i: 0, -3, -6, -9

Example 4: Print the numeric code of the characters from 15 to 85.

```
for (int i = 15; i <= 85; i++)
{
  char c = (char)i;
  System.out.println("i = " + i + " -> c = " + c);
}
```

Example 5: Print the squares of the integers between 1 and 10 using a for loop.

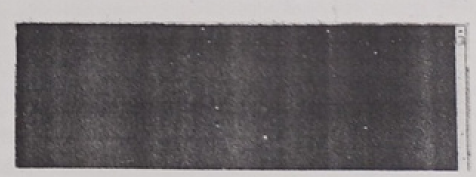
```
for (int i = 1; i <= 10; i++)
System.out.println(i * i);
```

Example 6: To find prime number between a given range.

```
class Prime1

{
  public static void main(String[] args)
  {
    //define limit
    int limit = 100;
    System.out.println("Prime numbers between 1 and " + limit);
    //loop through the numbers one by one
    for(int i=1; i < 100; i++)
      {
      boolean isPrime = true;
      //check to see if the number is prime
      for(int j=2; j < i ; j++)
        {
        if(i % j == 0)
        {
        isPrime = false;
        break;
        }
      }
      // print the number
      if(isPrime)
      System.out.print(i + " ");
    }
  }
}
```

Output:



3.2.4 Nested Loops

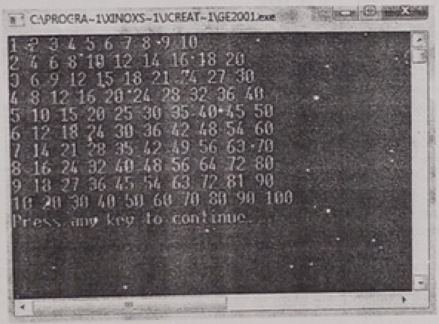
The body of a loop can contain itself a loop, called a nested loop. It is possible to nest an arbitrary number of loops.

The following example generates a multiplication table 2 through 9. Outer loop is responsible for generating a list of dividends, and inner loop will be responsible for generating lists of dividers for each individual number:

Example 7: Print out the multiplication table.

```
public class Table
{
  static final int NMAX = 10;
  public static void main (String[] args)
  {
    int row, column;
    for (row = 1; row <= NMAX; row++)
    {
      for (column = 1; column <= NMAX; column++)
        System.out.print(row * column + " ");
    System.out.println();
    }
  }
}
```

Output:



```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
Press any key to continue...
```

Example 8: This Java Pyramid example shows how to generate pyramid or triangle like given below using for loop.

```
*
* *
* * *
* * * *
* * * * *
```

```
class Pattern1{

  public static void main(String[] args) {

    for(int i=1; i<= 5 ;i++){

      for(int j=0; j < i; j++){
        System.out.print("*");
      }

      //generate a new line
      System.out.println("");
    }
  }
}
```

Output

```
*
* *
* * *
* * * *
* * * * *
```

Example 9: This Java Pyramid example shows how to generate pyramid or triangle like given below using for loop.

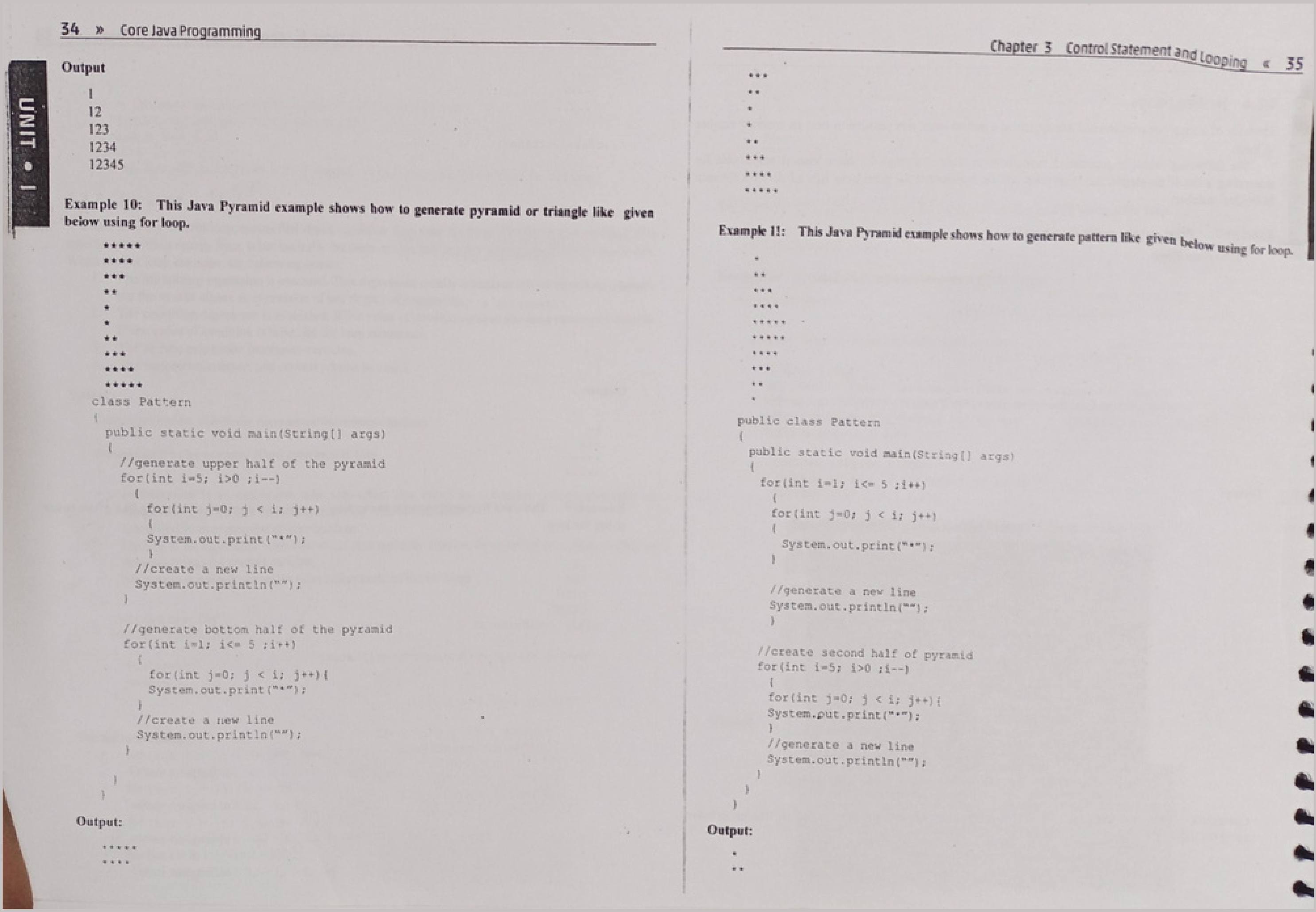
```
  1
  12
  123
  1234
  12345
class JavaPyramid4 {

  public static void main(String[] args)
  {

    for(int i=1; i<= 5 ;i++)
    {
      for(int j=0; j < i; j++){
      System.out.print(j+1);
      }

     System.out.println("");
    }

  }
}
```



Output

```
1
12
123
1234
12345
```

Example 10: This Java Pyramid example shows how to generate pyramid or triangle like given below using for loop.

```
*****
****
***
**
*
*
**
***
****
*****
```

```
class Pattern
{
  public static void main(String[] args)
  {
    //generate upper half of the pyramid
    for(int i=5; i>0 ;i--)
      {
        for(int j=0; j < i; j++)
        {
        System.out.print("*");
        }
      //create a new line
      System.out.println("");
    }

    //generate bottom half of the pyramid
    for(int i=1; i<= 5 ;i++)
      {
        for(int j=0; j < i; j++){
        System.out.print("*");
      }
      //create a new line
      System.out.println("");
    }

  }
}
```

Output:

```
*****
****
```

```
***
**
*
*
**
***
****
*****
```

Example 11: This Java Pyramid example shows how to generate pattern like given below using for loop.

```
*
**
***
****
*****
*****
****
***
**
*
```

```
public class Pattern
{
  public static void main(String[] args)
  {
    for(int i=1; i<= 5 ;i++)
      {
      for(int j=0; j < i; j++)
      {
        System.out.print("*");
      }

      //generate a new line
      System.out.println("");
      }

    //create second half of pyramid
    for(int i=5; i>0 ;i--)
      {
      for(int j=0; j < i; j++){
      System.out.print("*");
      }
      //generate a new line
      System.out.println("");
    }
  }
}
```

Output:

```
*
**
```

```
***
****
*****
*****
****
***
**
*
```

Example 12: This Java Pyramid example shows how to generate Pattern like given below using for loop.

```
12345
1234
123
12
1
public class Pattern {

  public static void main(String[] args) {

    for(int i=5; i>0 ;i--){

      for(int j=0; j < i; j++){
        System.out.print(j+1);
      }

      System.out.println("");
    }

  }
}
```

Output:

```
12345
1234
123
12
1
```

3.3 Jump Statements

[illegible]

3.3.1 The Break Statement

The *break* statement is used to exit a loop early, breaking out of the enclosing curly braces. When break statement is encountered control goes out of loop without checking the condition of loop and without executed further statements of loop.

Syntax

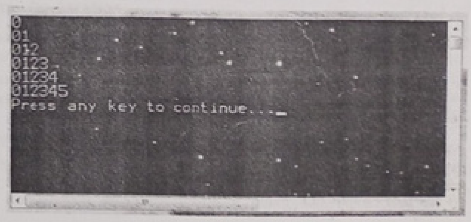
```
boolean finished = false;
while (condition && !finished)
{
  statements-1
  if (break-condition )
  {
  finished = true;
  break;
  }
  else
  {
    statements-2
  }
}
```

Example 13:

```
class Label1
{
  public static void main(String[] s)
  {

    for (int i = 0; i <6;i++)
    {
    for (int j= 0; j <6;j--)
      {
        if (j>i)
        break;
        System.out.print(""+j);
      }
      System.out.println();
    }
  }
}
```

Output:



```
0
01
012
0123
01234
012345
Press any key to continue..._
```

3.3.2 The Continue Statement

The *continue* statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

When a *continue* statement is encountered, program flow will move to the loop check expression immediately and if condition remain true then it start next iteration otherwise control comes out of the loop.

Example 14: Print out the odd numbers between 0 and 100.

```
for (int i = 0; i <= 100; i++)
{
 if (i % 2 == 0)
 continue;
 System.out.println(i);
}
```

3.3.3 The Return Statement

The return statement is also jump statement. The return statement exits from the current method, and control flow returns to where the method was invoked. The return statement has two forms, one that returns a value, and one that doesn't. To return a value, simply put the value (or an expression that calculates the value) after the return keyword.

E.g. return count;

The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.

return;

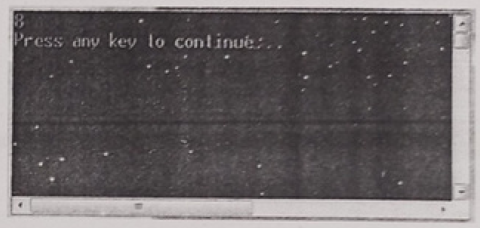
3.3.4 Labeled Break an Labeled Continue

In java break and continue keywords are working with label also. As use of goto is not a better habit as it increase the program complexity and decrease the readability of program.

Example 15: Program using labeled break.

```
class Label2
{
 public static void main(String[] s)
 {
  int x = 9;
  int y = 6;
  for (int z = 0; z <6;z++,y--)
  {
   if (x>2)x--;
   label:
   if (x>5)
   {
    while(true)
    {
    System.out.println(x);
    --x;
    break label;
    }
   }
   x--;
  }
 }
}
```

Output:



```
8
Press any key to continue...
```

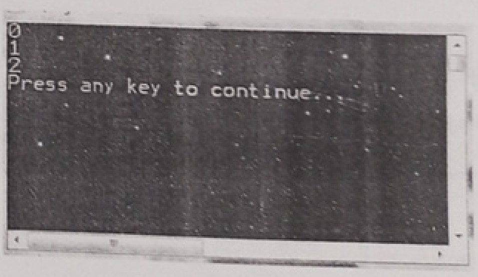
Example 16: Program using labeled break.

```
class Label2
{
 public static void main(String[] s)
 {

  abc:  for (int i = 0; i <6;i++)
  {
   for (int j= 0; j <6;j++)
   {

    if (j>2)
    break abc;
    System.out.println(""+j);
   }
  }
 }
}
```

Output:



```
0
1
2
Press any key to continue...
```