APPENDIX A

Java Language Reference

Important Keywords

Data-declaration keywords:				
`byte`	`int`	`float`	`char`	`double`
Loop keywords:				
`do`	`while`	`for`	`break`	`continue`
Conditional keywords:				
`if`	`else`	`switch`		
Exception keywords:				
`throw`	`try`	`catch`		
Structure keywords:				
`class`	`extends`	`interface`	`implements`	
Access keywords:				
`public`	`private`	`protected`		

Specifying Character Literals

Description or Escape Sequence	*Sequence*	*Output*
any character	'y'	y
backspace BS	'\b'	back space
Horizontal tab HT	'\t'	tab
line feed LF	'\n'	linefeed
form feed FF	'\f'	form feed
carriage return CR	'r'	carriage return
double quote	'\"'	``
single quote	'\''	`
backslash	'\\'	\
octal bit pattern	'0ddd'	(octal value of ddd)
hex bit pattern	'0xdd'	(hex value of dd)
Unicode character	'\dddd'	(actual Unicode character of dddd)

Arithmetic Operators

Operator	Operation	Example
+	Addition	x + y
–	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

Assignment Operators

Operator	Operation	Example	Meaning
+ =	add to current variable	x + = y	x = x + y
– =	subtract from current variable	x – = y	x = x – y
* =	multiply by current variable	x * = y	x = x * y
/ =	divide by current variable	x/ = y	x = x/y

Increment and Decrement Operators

Operator	Operation	Example	Meaning
+ +	increment by 1	x + +	x = x + 1
– –	decrement by 1	x – –	x = x – 1

Comparison Operators (return true or false)

Operator	Operation	Example	Meaning
= =	Equal	x = = y	Is x equal to y?
! =	Not equal	x ! = y	Is x not equal to y?
<	Less than	x < y	Is x less than y?
>	Greater than	x > y	Is x greater than y?
< =	Less than or equal to	x < = y	Is x less than or equal to y?
> =	Greater than or equal to	x > = y	Is x greater than or equal to y?

Bitwise Operators

Operator	Operation
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
< <	Left shift
> >	Right shift
> > >	Zero fill right shift
~	Bitwise complement
< < =	Left shift assignment
> > =	Right shift assignment
> >> =	Zero fill right shift assignment
x & = y	AND assignment
x | = y	OR assignment
x ^ = y	NOT assignment

Comment Indicators

Start	Text	End Comment
/*	text	*/
/**	text	*/
//	text	(everything to the end of the line is ignored by the compiler)

Primitive Data Type Keywords

```
boolean   char   byte   short   int   long   float   double
```

Integer Data Type Ranges

Type	Length	Minimum Value	Maximum Value
byte	8 bits	–128	127
Short	16 bits	–32768	32767
Int	32 bits	–2147483648	2147483647
long	64 bits	–9223372036854775808	9223372036854775807

Unary operators

Operator	Operation
–	Unary negation
~	Bitwise complement
++	Increment
--	Decrement
!	Not

Operator Precedence

++	--	!	~	instanceof	*
/	%	+	–	<<	>>
>>>	<	>	<=	>=	==
!=	&	^	&&	||	?:
=	op=				

Control Statements

Statement	Example
A simple `if` statement	`if (booleanTest)` `  callfunction ( );`

A multiline `if` statement	`if (booleanTest)`
	`{`
	`  // set of statements`
	`}`

The `if...else` statement	`if (booleanTest)`
	`{`
	`  // True block statements`
	`else`
	`{`
	`  // False block statements`
	`}`

(Contd)

Statement	Example

The `while` statement	`while (booleanTest)`
	`{`
	`  // Loop statements`
	`}`

The `do...while` loop	`do`
	`{`
	`  // Loop statements`
	`}`
	`while (booleanTest);`

The `switch` statement	`switch (expression)`
	`{`
	`  case FirstCase:`
	`  // First set of statements`
	`  break ;`
	`  case SecondCase:`
	`  // Second set of statements`
	`  break ;`
	`  case ThirdCase :`
	`  // Third set of statements`
	`  break;`
	`  default :`
	`  // Default statement`
	`  break ;`
	`}`

The `for` loop	`for (initialization; condition; increment) statement;`

Defining Classes

The basic structure of defining a class is as follows:

```
Scope class ClassName [extends class]
{
    // Class implementation statements
}
```

When declaring the scope of the class, we have several options to control how other classes can access this class:

`public`	The class can be used by code outside of the file. Only one class in a file may have this scope. The file must be named with the class name followed by further four-letter .java extension.
`private`	The class can only be used within a file.
`abstract`	The class cannot be used by itself and must be subclassed.
`final`	The class cannot be used by a subclass.
`Synchronizable`	Instances of this class can be made arguments.

If a scope modifier is not used, the class is only accessible within the current file.

Defining Methods

A *method* is the code that acts on data inside a class and is always declared inside the class declaration. A method has the following syntax:

```
Scope ReturnType methodName (arguments)
{
    // Method implementation statements
}
```

The scope allows the programmer to control access to methods and can be one of the following:

`public`	The method is accessible by any system object.
`protected`	The method is only accessible by subclasses and the class in which it is declared.
`private`	The method is accessible only within current class.
`final`	The method cannot be overridden by any subclass.
`Static`	The method is shared by all instances of the class.

If a method is not given a scope, it is only accessible within the scope of the current file. We can also use these scope operators when declaring variables.

Exception Handling

An exception has two parts: signalling an exception and setting up an exception handler. To signal an exception, use the **try** keyword. To set up an exception handler, we use the **catch** keyword. We use the **finally** keyword to specify a block of statements that will execute no matter what. To tell the system that an error has occurred, use the **throw** keyword.

```
try
{
        // Try this block of code and throw exception
{
catch (Exception e)
{
        // Handle error
}
finally
{
        // Executed no matter what happens
}
```

General Applet Construction

A minimal Java Applet has the following construction:

```
/*
*   JavaApplet.java - Sample Applet *
*/
import java.applet.*;
import java.awt.Graphics;
public class JavaApplet extends java.applet.Applet
{
    public void init (  )
    {
        // Called first time applet is executed
    }
    public void start (  )
    {
        // Called after init( ) and whenever Web page is revisited
    }
    public void stop (  )
    {
        // Called when Web page disappears
    }
    public void destroy (  )
    {
        //Called when applet is being removed from memory
    }
    public void paint (Graphics g)
    {
        g.drawstring ("Goodbye !", 100, 100) ;
```

APPENDIX **B**

Java Keywords

This appendix lists the keywords in Java. They are grouped according to their meaning/function.

Group	*Keyword*	*Meaning/Function*
Class Organization	`package`	specifies the class in a particular source file should belong to the named package.
	`import`	requests the named class or classes be imported into the current application.
Class Definition	`interface`	defines global data and method signatures that can be shared among classes.
	`class`	defines a collection of related data behaviour.
	`extends`	indicates which class to subclass.
	`implements`	indicates the interface for which a new class will supply methods.
Keywords for Classes and Variables	`abstract`	specifies the class cannot be instantiated directly.
	`public`	means the class, method, or variable can be accessed from anywhere.
	`private`	means only the class defining the method or variable can access it.
	`protected`	means only the defining class and its subclasses can access the method or variable.
	`static`	specifies a class method or variable.
	`synchronized`	indicates only one object or class can access this variable or method at a time.
	`volatile`	tells the compiler this variable may change asynchronously due to threads.
	`final`	means this variable or method cannot be changed by subclasses.
	`const`	means this variable cannot be changed.
	`native`	links a method to native code.
Simple Data Types	`long`	is a 64-bit integer value.
	`int`	is a 32-bit integer value.
	`short`	is a 16-bit integer value.

(*Contd*)

(*Contd*)

Group	Keyword	Meaning/Function
Values and Variables	`byte`	is a 8-bit integer value.
	`double`	is a 64-bit floating-point value.
	`float`	is a 32-bit floating-point value.
	`char`	is a 16-bit Unicode character.
	`boolean`	is a true or false value.
	`void`	indicates a method does not return a value.
	`false`	is a Boolean value.
	`true`	is a Boolean value.
	`this`	refers to the current instance in an instance method.
	`super`	refers to the immediate superclass in an instance method.
	`null`	represents a nonexistent instance.
Exception Handling	`throw`	throws an exception
	`throws`	throws an exception.
	`try`	marks a stack so that if an exception is thrown, it will unwind to this point.
	`catch`	catches an exception.
	`finally`	says execute this block of code regardless of exception error handling flow.
Instance Creating and Testing	`new`	creates new instances.
	`instanceof`	tests whether an instance derives from a particular class or interface.
Control Flow	`switch`	tests a variable.
	`case`	executes a particular block of code according to the value tested in the switch.
	`default`	means the default block of code executes if no matching case statement was found.
	`break`	breaks out of a particular block of code.
	`continue`	continues with the next iteration of a loop.
	`goto`	directs control to a specified place.
	`return`	returns from a method, optionally passing back a value.
	`do`	performs some statement or set of statements.
	`if`	tests for a condition and performs some action if true.
	`else`	performs some action if the above test was false.
	`for`	signifies iteration.
	`while`	performs some action while a condition is true.

- **Keywords not available from C**
  `auto, enum, extern, register, signed, sizeof, struct, typedef, union, unsigned.`
- **Keywords not available from C++**
  `delete, friend, inline, mutable, template, using, virtual.`

APPENDIX

C

Differences Between Java and C/C++

C.1 DATA TYPES

- All Java primitive data types (char, int, short, long, byte, float, double and boolean) have specified sizes and behaviour that are machine-independent.
- Conditional expressions can only be Boolean, not integral.
- Casting between data types is much more controlled in Java. Automatic conversion occurs only when there is no loss of information. All other casts must be explicit.
- Java supports special methods to convert values between class objects and primitive types.
- Composite data types are accomplished in Java using only classes. Structures and unions are not supported.
- Java does not support **typedef** keyword.
- All non-primitive types can only be created using **new** operator.
- Java does not define the type modifiers **auto, extern, registor, signed,** and **unsigned**.

C.2 POINTERS

- Java does not support pointers. Similar functionality is accomplished by using implicit references to objects. Pointer arithmetic is not possible in Java.

C.3 OPERATORS

- Java adds a new right shift operator > > > which inserts zeros at the top end.
- The + operator can be used to concatenate strings.
- Operators overloading is not possible in Java.
- The , operator of C has been deleted.

- Java adds another operator **instanceof** to identify objects.
- The modulo division may be applied to float values in Java which is not permitted in C/C++.

C.4 FUNCTIONS AND METHODS

- All functions are defined in the body of the class. There are no independent functions.
- The functions defined inside a class are known as methods.
- Although function overloading in Java works virtually identical to C++ function overloading, there are no default arguments to functions.
- No inline functions in Java.
- Java requires that methods with no arguments must be declared with empty parenthesis, (not with **void** keyword).

C.5 PREPROCESSOR

- Java does not have a preprocessor, and as such, does not support **#define** or macros.
- Constants can be created using the **final** modifier when declaring class and instance variables.
- Java programs do not use header files.

C.6 CLASSES

- Class definitions take the similar form in Java as in C++, but there is no closing semicolon.
- There is no scope resolution operator :: in Java.
- No forward references of classes are necessary in Java.
- No destructors in Java.
- Java has no templates.
- No nested classes in Java.
- Inheritance in Java has the same effect as in C++, but the syntax is different.
- Java does not provide direct support for multiple inheritance. We can accomplish multiple inheritance by using interfaces.
- Access specifiers (public, private, protected and private protected) are placed on each definition for each member of a class.
- A class in Java can have an access specifier to determine whether it is visible outside the file.
- There is no **virtual** keyword in Java. All non-static methods always use dynamic binding.
- Initialization of primitive class data member is guaranteed in Java. We can initialize them directly when we define them in the class, or we can do it in the constructor.
- We need not externally define storage for **static** members like we do in C++.

C.7 STRINGS

- Strings in C and C++ are arrays of characters, terminated by a null character. But strings in Java are objects. They are not terminated by a null. Therefore, strings are treated differently in C++ and Java.
- Strings can be concatenated using + operator.

C.8 ARRAYS

- Arrays are quite different in Java. Array boundaries are strictly enforced. Attempting to read past the end of an array produces an error.
- One array can be assigned to another in Java.
- Java does not support multidimensional arrays as in C and C++. However, it is possible to create arrays of arrays to represent multidimensional arrays.

C.9 CONTROL FLOW

- The test expressions for control flow constructs return a Boolean value (true or false) in Java. In C and C++, they return an integer value.
- The control variable declared in **for** loop is not available after the loop is exited in Java.

C.10 COMMAND-LINE ARGUMENTS

- The command line arguments passed from the system into a Java program differ in a couple of ways compared to that of C++ program.
- In C and C++, two arguments are passed. One specifies the number of arguments and the other is a pointer to an array of characters containing the actual arguments. In Java, a single argument containing an array of strings is passed.
- The first element in the arguments vector in C and C++ is the name of the program itself. In Java, we do not pass the name of the program as an argument. We already know the name of the program because it is the same name as the class.

C.11 OTHER DIFFERENCES

- Java supports multithreading.
- Java supports automatic garbage collection and makes a lot of programming problems simply vanish.
- The destructor function is replaced with a finalize function.
- Exception handling in Java is different because there are no destructors. A **finally** clause is always executed to perform necessary cleanup.
- Java has built-in support for comment documentation, so the source code file can also contain its own documentation.

APPENDIX

D

Bit-level Programming

D.1 INTRODUCTION

One of the unique features of Java language as compared to other high-level languages is that it allows direct manipulation of individual bits within a word. Bit-level manipulations are used in setting a particular bit or group of bits to 1 or 0. They are also used to perform certain numerical computations faster. As pointed out in Chapter 5, Java supports the following operators:

1. Bitwise logical operators
2. Bitwise shift operators
3. One's complement operator

All these operators work only on integer type operands.

D.2 BITWISE LOGICAL OPERATORS

There are three logical Bitwise operators. They are:

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise *exclusive* OR (^)

These are binary operators and require two integer-type operands. These operators work on their operands bit by bit starting from the least significant (i.e. the rightmost) bit, setting each bit in the result as shown in Table D.1.

Table D.1 *Java Milestones*

op1	*op2*	*op1 & op2*	*op1 | op2*	*op1 ^ op2*
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Bitwise AND

The bitwise AND operator is represented by a single ampersand (&) and is surrounded on both sides by integer expressions. The result of ANDing operation is 1 if both the bits have a value of 1; otherwise it is 0. Let us consider two variables **x** and **y** whose values are 13 and 25. The binary representation of these two variables are

```
x → 0000 0000 0000 1101
y → 0000 0000 0001 1001
```

If we execute statement

```
z = x & y;
```

then the result would be:

```
z → 0000 0000 0000 1001
```

Although the resulting bit pattern represents the decimal number 9, there is no apparent connection between the decimal values of these three variables. Program D.1 shows how to use the bitwise operators.

Program D.1 *Demonstration of bitwise operators*

```
Class Bitwise
{
   public static void main (String args[])
   {
      int a=13, b=25;
      System.out.println ("a = " + a);
      System.out.println ("b = " + b);
      System.out.println ("a & b = " + (a & b) );
      System.out.println ("a | b = " + (a | b) );
      System.out.println ("a ^ b = " + (a ^ b) );
   }
}
```

The output would be:

```
a = 13
b = 25
a & b = 9
a | b = 29
a ^ b = 20
```

Bitwise ANDing is often used to test whether a particular bit is 1 or 0. For example, the following program tests whether the fourth bit of the variable **flag** is 1 or 0.

```
Class Bit1
{
    Static final TEST = 8;                    /* represents 00....01000 */
    public static void main (String args[])
    {
        int flag;
        ..........
        ..........
        if ( (flag & TEST) ! = 0)             /* test 4th bit */
        {
```

```
                System.out.println ("Fourth bit is set \n") ;
            }
            ..........
            ..........
        }
    }
```

Note that the bitwise logical operators have lower precedence than the relational operators and therefore additional parentheses are necessary as shown above.

The following program tests whether a given number is odd or even.

```
Class Bit2
{
    public static void main (String args [])
    {
        int test = 1;
        int number;
        // Input a number here
            ..........
            ..........
        while (number ! = -1)
        {
            if (number & test)
                System.out.println ("Number is odd\n\n");
        else
                System.out.println ("Number is even\n\n");
        // Input a number here
        ..........
        ..........
    }
  }
}
```

Output:

```
Input a number
20
Number is even
Input a number
9
Number is odd
Input a number
-1
```

Bitwise OR

The bitwise OR is represented by the symbol | (vertical bar) and is surrounded by two integer operands. The result of OR operation is 1 if *at least* one of the bits has a value of 1; otherwise it is zero. Consider the variables x and y discussed above.

```
x       →   0000  0000  0000  1101
y       →   0000  0000  0001  1001
x | y   →   0000  0000  0001  1101
```

The bitwise inclusion OR operation is often used to set a particular bit to 1 in a flag. Example:

```
Class Bit3
{
    final static SET = 8;
    public static void main (String args[])
    {
        int flag;
        ..........
        ..........
        flag = flag | SET;
        if ((flag & SET) != 0)
        {
            System.out.println ("flag is set \n");
        }
        ..........
        ..........
    }
}
```

The statement

```
flag = flag | SET;
```

causes the fourth bit of flag to set 1 if it is 0 and does not change it if it is already 1.

Bitwise Exclusive OR

The bitwise *exclusive* OR is represented by the symbol ^. The result of exclusive OR is 1 if *only one* of the bits is 1; otherwise it is 0. Consider again the same variables **x** and **y** discussed above.

```
x      →   0000 0000 0000 1101
y      →   0000 0000 0001 1001
x^y    →   0000 0000 0001 1101
```

D.3 BITWISE SHIFT OPERATORS

The shift operators are used to move bit patterns either to the left or to the right. The shift operators are represented by the symbols < and > and are used in the following form:

```
Left shift:  op << n
Right shift:      op >> n
```

op is the integer expression that is to be shifted and *n* is the number of bit positions to be shifted.

The left-shift operation causes all the bits in the operand *op* to be shifted to the left by *n* positions. The leftmost *n* bits in the original bit pattern will be lost and rightmost *n* bits positions that are vacated will be filled with 0s.

Similarly, the right-shift operation causes all the bits in the operand *op* to be shifted to the right by *n* positions. The rightmost *n* bits will be lost. The leftmost *n* bit positions that are vacated will be filled with zero, if the *op* is a *positive integer*. If the variable to be shifted is *negative*, then the operation preserves the high-order bit of 1 and shifts only the lower 31 bits to the right.

Both the operands *op* and *n* can be constants or variables. There are two restrictions on the value of *n*. It may not be negative and it may not exceed the number of bits used to represent the left operand *op*.

Let us suppose **x** is a positive integer whose bit pattern is

```
0100    1001    1100    1011
```

then,

```
                                          Vacated
                                          positions
                                          ↙
x << 3  =  0100  1110  0101  1000
x >> 3  =  0000  1001  0011  1001
             ↑
          Vacated
          positions
```

Shift operators are often used for multiplication and division by powers of two.
Consider the following statement:

```
x = y << 1;
```

This statement shifts one bit to the left in **y** and then the result is assigned to **x**. The decimal value of x will be the value of y multiplied by 2. Similarly, the statement

```
x = y >> 1;
```

shifts **y** one bit to the right and assigns the result to **x**. In this case, the value of **x** will be the value of y divided by 2.

Java supports another shift operator >>> known as zero-fill-right-shift operator. When dealing with positive numbers, there is no difference between this operator and the right-shift operator. They both shift zeros into the upper bits of a number. The difference arises when dealing with negative numbers. Note that negative numbers have the high-order bit set to 1. The right-shift operator preserves the high-order bit as 1. The zero-fill-right-shift operator shifts zeros into all the upper bits, including the high-order bit, thus making a negative number into positive. Program D.2 demonstrates the use of shift operators.

Program D.2 *Demonstration of Shift operators*

```
Class Shift
{
    public static void main (String args[])
    {
        int a=8, b=-8;
        System.out.println ("a = " + a + " b = " + b);
        System.out.println ("a >> 2 = " + (a >> 2));
        System.out.println ("a << 1 = " + (a << 1));
        System.out.println ("a >>> 1 = " + (a >>> 1));
        System.out.println ("b >> 1 = " + (b >> 1));
        System.out.println ("b >>> 1 = " + (b >>> 1));
    }
}
```

The output would be:

```
a = 8     b = - 8
a >> 2      = 2
a << 1      = 16
a >>> 1     = 4
b >> 1      = - 4
b >>> 1     = 2147483644
```

D.4 BITWISE COMPLEMENT OPERATORS

The complement operator ~ (also called the one's complement operator) is an unary operator and inverts all the bits represented by its operand. That is, 0s become 1s and 1s become zero. Example:

```
x    =    1001 0110 1100 1011
~x   =    0110 1001 0011 0100
```

This operator is often combined with the bitwise AND operator to turn off a particular bit. For example, the statements

```
x    =  8;          /* 0000 0000 00000 1000 */
flag =  flag & ~x;
```

would turn off the fourth bit in the variable **flag.**

APPENDIX **E**

Java API Packages

Java API is implemented as packages, which contain groups of related classes. Along with classes, they also include interfaces, exception definitions and error definitions. Java API is composed of a large number of packages. The most commonly used packages are:

Stand-alone Application Programming

1. java.lang
2. java.util
3. java.io

Applet and Network Programming

4. java.awt
5. java.applet
6. java.net

This appendix lists the frequently used interfaces and classes contained in the above packages.

Table E.1 *Java.lang Package*

Interfaces	
`Cloneable`	Interface indicating that an object may be copied or cloned
`Runnable`	Methods for classes that want to run as threads
Classes	
`Boolean`	Object wrapper for boolean values
`Byte`	Object wrapper for byte values
`Character`	Object wrapper for char values
`Class`	Run-time representations of classes
`ClassLoader`	Abstract behaviour for handling loading of classes

(Contd)

`Compiler`	System class that gives access to the Java compiler
`Double`	Object wrapper for double values
`Float`	Object wrapper for float values
`Integer`	Object wrapper for int values
`Long`	Object wrapper for long values
`Math`	Utility class for math operations
`Number`	Abstract superclass of all number classes (Integer, Float, and so on)
`Object`	Generic object class, at top of inheritance hierarchy
`Process`	Abstract behaviour for processes such as those spawned using methods in the System class
`Runtime`	Access to the Java runtime
`SecurityManager`	Abstract behaviour for implementing security policies
`String`	Character strings
`StringBuffer`	Mutable strings
`System`	Access to Java's system-level behaviour, provided in a platform independent way
`Thread`	Methods for managing threads and classes that run in threads
`ThreadDeath`	Class of object thrown when a thread is asynchronously terminated
`ThreadGroup`	A group of threads
`Throwable`	Generic exception class; all objects thrown must be a Throwable

Table E.2 *Java.util Package*

Interfaces	
`Enumeration`	Methods for enumerating sets of values
`Observer`	Methods for enabling classes to be Observable objects
Classes	
`BitSet`	A set of bits
`Date`	The current system date, as well as methods for generating and parsing dates
`Dictionary`	An abstract class that maps between keys and values (superclass of HashTable)
`Hashtable`	A hash table
`Observable`	An abstract class for observable objects
`Properties`	A hash table that contains behaviour for setting and retrieving persistent properties of the system or a class
`Random`	Utilities for generating random numbers
`Stack`	A stack (a last-in-first-out queue)
`StringTokenizer`	Utilities for splitting strings into individual "token"
`Vector`	A growable array of Objects

Table E.3 *Java.io Package*

Interfaces	
`DataInput`	Methods for reading machine-independent typed input streams
`DataOutput`	Methods for writing machine-independent typed output streams

(Contd)

FilenameFilter	Methods for filtering file names
Classes	
BufferedInputStream	A buffered input stream
BufferedOutputStream	A buffered output stream
ByteArrayInputStream	An input stream from a byte array
ByteArrayOutputStream	An output stream to a byte array
DataInputStream	Enables you to read primitive Java types (ints, chars, booleans, and so on) from a stream in a machine-independent way
DataOutputStream	Enables you to write primitive Java data types (ints, chars, booleans, and so on) to a stream in a machine-component way
File	Represents a file on the host's file system
FileDescriptor	Holds onto the UNIX-like file descriptor of a file or socket
FileInputStream	An input stream from a file, constructed using a filename of descriptor
FileOutputStream	An output stream to a file, constructed using a filename or descriptor
FilterInputStream	Abstract class which provides a filter for input streams (and for adding stream functionality such as buffering)
FilterOutputStream	Abstract class which provides a filter for output streams (and for adding stream functionality such as buffering)
Input Stream	An abstract class representing an input stream of bytes; the parent of all input streams in this package
LineNumberInputStream	An input stream that keeps track of line numbers
OutputStream	An abstract class representing an output stream of bytes; the parent of all output stream in this package
PipedInputStream	A piped input stream, which should be connected to a PipedOutputStream to be useful
PipedOutputStream	A piped output stream, which should be connected to a PipedInputStream to be useful (together they provide safe communication between threads)
PrintStream	An output stream for printing (used by System.out.println(...))
PushbackInputStream	An input stream with a 1-byte push back buffer
RandomAccessFile	Provides random access to a file, constructed from filenames, descriptors or objects
SequenceInputStream	Converts a sequence of input streams into a single input stream
StreamTokenizer	Converts an input stream into a series of individual tokens
StringBufferInputStream	An input stream from a String object

Table E.4 *Java.awt Package*

Interfaces	
LayoutManager	Methods for laying out containers
MenuContainer	Methods for menu-related containers

(Contd)

Classes	
BorderLayout	A layout manager for arranging items in border formation
Button	A UI pushbutton
Canvas	A canvas for drawing and performing other graphics operations
CardLayout	A layout manager for HyperCard-like metaphors
Checkbox	A checkbox
CheckboxGroup	A group of exclusive checkboxes (radio buttons)
CheckboxMenuItem	A toggle menu item
Choice	A popup menu of choices
Color	An abstract representation of a color
Component	The abstract generic class for all UI components
Container	Abstract behaviour for a component that can hold other components or containers
Dialog	A window for brief interactions with users
Dimension	An object representing width and height
Event	An object representing events caused by the system or based on user input
FileDialog	A dialog for getting filenames from the local file system
FlowLayout	A layout manager that lays out objects from left to right in rows
Font	An abstract representation of a font
FontMetrics	Abstract class for holding information about a specific font's character shapes and height and width information
Frame	A top-level window with a title
Graphics	Abstract behaviour for representing a graphics context, and for drawing and painting shapes and objects
GridBagConstraints	Constraints for components laid out using GridBagLayout
GridBagLayout	A layout manager that aligns components horizontally and vertically based on their values from GridBagConstraints
GridLayout	A layout manager with rows and columns; elements are added to each cell in the grid
Image	An abstract representation of a bitmap image
Insets	Distances from the outer border of the window; used to layout components
Label	A text label for UI components
List	A scrolling list
MediaTracker	A way to keep track of the status of media objects being loaded over the Net
Menu	A menu, which can contain menu items and is a container on a menubar
Menubar	A menubar (container for menus)
MenuComponent	The abstract superclass of all menu elements
MenuItem	An individual menu item
Panel	A container that is displayed
Point	An object representing a point (x and y coordinates)
Polygon	An object representing a set of points
Rectangle	An object representing a rectangle (x and y coordinates for the top corner, plus width and height)
Scrollbar	A UI scrollbar object

(Contd)

TextArea	A multilane, scrollable, editable text field
TextComponent	The supeclass of all editable text components
TextField	A fixed-size editable text field
Toolkit	Abstract behaviour for binding the abstract AWT classes to a platform-specific toolkit implementation
Window	A top-level window, and the superclass of the Frame and Dialog classes

Table E.5 *Java.awt.image Package*

Interfaces	
ImageConsumer	Methods for receiving image created by an ImageProducer
ImageObserver	Methods to track the loading and construction of an image
ImageProducer	Methods for producing image data received by an ImageConsumer
Classes	
ColorModel	An abstract class for managing color information for images
CropImageFilter	A filter for cropping images to a particular size
DirectColorModel	A specific color model for managing and translating pixel color values
FilteredImageSource	An ImageProducer that takes an image and an ImageFilter object, and produces an image for an ImageConsumer
ImageFilter	A filter that takes image data from an ImageProducer, modifies it in some way, and hands it off to an ImageConsumer
IndexColorModel	A specific color model for managing and translating color values in a fixed-color map
MemoryImageSource	An image producer that gets its image from memory; used after constructing an image by hand
PixelGrabber	An ImageConsumer that retrieves a subset of the pixels in an image
RGBImageFilter	Abstract behaviour for a filter that modifies the RGB values of pixels in RGB images

Table E.6 *Java.applet Package*

Interfaces	
AppletContext	Methods to refer to applet's context
AppletStub	Methods to implement applet viewers
AudioClip	Methods to play audio files
Classes	
Applet	The base applet class

Table E.7 *Java.net Package*

Interfaces	
ContentHandler Factory	Methods for creating ContentHandler objects
SocketImplFactory	Methods for creating socket implementations (instance of the SocketImpl class)
URLStreamHandlerFactory	Methods for creating URLStreamHandler objects

(Contd)

Classes	
`ContentHandler`	Abstract behaviour for reading data from a URL connection and constructing the appropriate local object, based on MIME type
`DatagramPacket`	A datagram packet (UDP)
`DatagramSocket`	A datagram socket
`InetAddress`	An object representation of an Internet host (host name, IP address)
`ServerSocket`	A sever-side socket
`Socket`	A socket
`SocketImpl`	An abstract class for specific socket implementations
`URL`	An object representation of a URL
`URLConnection`	Abstract behaviour for a socket that can handle various Web-based protocols (http, ftp, and so on)
`URLEncoder`	Turns strings into x-www-form-urlencoded format
`URLStreamHandler`	Abstract class for managing streams to object referenced by URLs

APPENDIX F

Java Classes and Their Packages

This appendix lists the frequently used classes in alphabetical order and indicates in which package a given class is defined. It also lists the classes that extend them.

Class	*Package*	*Subclasses*
AbstractMethodError	`java.lang`	Nil
AppletContext	`java.applet`	Nil
AppletStub	`java.applet`	Nil
Applet	`java.applet`	Nil
ArithmeticException	`java.lang`	Nil
ArrayIndexOutofBoundsException	`java.lang`	Nil
ArrayStoreException	`java.lang`	Nil
AudioClip	`java.applet`	Nil
AWTError	`java.awt`	Nil
AWTException	`java.awt`	Nil
BitSet	`java.util`	Nil
Boolean	`java.lang`	Nil
BorderLayout	`java.awt`	Nil
BufferedInputStream	`java.io`	Nil
BufferedOutputStream	`java.io`	Nil
ButtonPeer	`java.awt.peer`	Nil
Button	`java.awt`	Nil
ByteArrayInputStream	`java.io`	Nil
ByteArrayOutputStream	`java.io`	Nil
CanvasPeer	`java.awt.peer`	Nil
Canvas	`java.awt`	Nil
CardLayout	`java.awt`	Nil
Character	`java.lang`	Nil

(Contd)

(Contd)

Class	Package	Subclasses
CheckboxGroup	java.awt	Nil
CheckboxMenuItemPeer	java.awt.peer	Nil
CheckboxMenuItem	java.awt	Nil
CheckboxPeer	java.awt.peer	Nil
Checkbox	java.awt	Nil
ChoicePeer	java.awt.peer	Nil
Choice	java.awt	Nil
ClassCastException	java.lang	Nil
ClassCircularityError	java.lang	Nil
ClassFormatError	java.lang	Nil
ClassLoader	java.lang	Nil
ClassNotFoundException	java.lang	Nil
Class	java.lang	Nil
Cloneable	java.lang	Nil
CloneNotSupportedException	java.lang	Nil
ColorModel	java.awt.image	DirectColorModel, IndexColorModel
Color	java.awt	Nil
Compiler	java.lang	Nil
ComponentPeer	java.awt.peer	ButtonPeer, CanvasPeer, CheckboxPeer, ChoicePeer, ContainerPeer, LabelPeer, ListPeer, ScrollbarPeer, TextComponentPeer
Component	java.awt	Button, Canvas, Checkbox, Choice, Container, Lable, List, Scrollbar, TextComponent
ContainerPeer	java.awt.peer	PanelPeer, WindowPeer
Container	java.awt	Panel, Window
ContentHandlerFactory	java.net	Nil
ContentHandler	java.net	Nil
CropImageFilter	java.awt.image	Nil
DatagramPacket	java.net	Nil
DatagramSocket	java.net	Nil
DataInputStream	java.io	Nil
DataInput	java.io	Nil
DataOutputStream	java.io	Nil
DataOutput	java.io	Nil
Date	java.util	Nil
DialogPeer	java.awt.peer	FileDialogPeer

(Contd)

(*Contd*)

Class	*Package*	*Subclasses*
Dialog	java.awt	FileDialog
Dictionary	java.util	Hashtable
Dimension	java.awt	Nil
DirectColorModel	java.awt.image	Nil
Double	java.lang	Nil
EmptyStackException	java.util	Nil
Enumeration	java.util	Nil
EOFException	java.io	Nil
Error	java.lang	AWTError, LinkageError, ThreadDeath, VirtualMachineError
Event	java.awt	Nil
Exception	java.lang	AWTException, ClassNotFoundException, CloneNotSupportedException, IllegalAccessException, InstantiationException, InterruptedException, IOException, NoSuchMethodException RuntimeException
FileDescriptor	java.io	Nil
FileDialogPeer	java.awt.peer	Nil
FileDialog	java.awt	Nil
FileInputStream	java.io	Nil
FilenameFilter	java.io	Nil
FileNotFoundException	java.io	Nil
FileOutputStream	java.io	Nil
File	java.io	Nil
FilteredImageSource	java.awt.image	Nil
FilterInputStream	java.io	BufferedInputStream, DataInputStream, LineNumberInputStream, PushbacklnputStream BufferedOutputStream, DataOutputStream, Print Stream
Float	java.io	Nil
FlowLayout	java.awt	Nil
FontMetrics	java.awt	Nil
Font	java.awt	Nil
FramePeer	java.awt.peer	Nil

(*Co*

(Contd)

Class	Package	Subclasses
Frame	java.awt	Nil
Graphics	java.awt	Nil
GridBagConstraints	java.awt	Nil
GridBagLayout	java.awt	Nil
GridLayout	java.awt	Nil
Hashtable	java.util	Properties
IllegalAccessError	java.lang	Nil
IllegalAccessException	java.lang	Nil
IllegalArgumentException	java.lang	IllegalThreadStateException, NumberFormatException
IllegalMonitorStateException	java.lang	Nil
IllegalThreadStateException	java.lang	Nil
ImageConsumer	java.awt.image	Nil
ImageFilter	java.awt.image	CropImageFilter, RGBImageFilter
ImageObserver	java.awt.image	Nil
ImageProducer	java.awt.image	Nil
Image	java.awt	Nil
IncompatibleClassChangeError	java.lang	AbstractMethodError, IllegalAccessError, InstantiationError, NoSuchFieldError, NoSuchMethodError
IndexColorModel	java.awt.image	Nil
IndexOutofBoundsException	java.lang	ArrayIndexOutOfBounds Exception, StringIndexOutOfBounds Exception
InetAddress	java.net	Nil
InputStream	java.io	ByteArrayInputStream, FileInputStream, FilterInputStream, PipedInputStream, SequenceInputStream, StringBufferInputStream
Insets	java.awt	Nil
InstantiationError	java.lang	Nil
InstantiationException	java.lang	Nil
Integer	java.lang	Nil
InternalErorr	java.lang	Nil
InterruptedException	java.lang	Nil

(*Contd*)

Class	*Package*	*Subclasses*
InterruptedIOException	java.io	NIl
IOException	java.io	EOFException, FileNotFoundException, InterruptedIOException, MalformedURLException, ProtocolException, SocketException, UnknownHostException, UnknownServiceException, UTFDataFormatException
LabelPeer	java.awt.peer	Nil
Label	java.awt	Nil
LayoutManager	java.awt	Nil
LineNumberInputStream	java.io	Nil
LinkageError	java.lang	ClassCircularityError ClassFormatError, IncompatibleClassChange Error, UnsatisfiedLinkError, VerifyError
ListPeer	java.awt.peer	Nil
ListPeer	java.awt	Nil
Long	java.lang	Nil
MalformedURLException	java.net	Nil
Math	java.lang	Nil
MediaTracker	java.awt	Nil
MemoryImageSource	java.awt.image	Nil
MenuBarPeer	java.awt.peer	Nil
MenuBar	java.awt	Nil
MenuComponentPeer	java.awt.peer	CheckboxMenuItemPeer, MenuPeer
MenuComponent	java.awt	MenuBar, MenuItem
MenuContainer	java.awt	Nil
MenuItemPeer	java.awt.peer	CheckboxMenuItemPeer, MenuPeer
MenuItem	java.awt	CheckboxMenuItem, Menu
MenuPeer	java.awt.peer	Nil
Menu	java.awt	Nil
NegativeArraySizeException	java.lang	Nil
NoClassDefFoundError	java.lang	Nil
NoSuchElementException	java.util	Nil

(Contd)

Class	Package	Subclasses
NoSuchFieldError	java.lang	Nil
NoSuchMethodError	java.lang	Nil
NoSuchMethodException	java.lang	Nil
NullPointerException	java.lang	Nil
NumberFormatException	java.lang	Nil
Number	java.lang	Nil
Object	java.lang	Double, Float, Integer, Long
		BitSet, Boolean, BorderLayout,
		CardLayout, Character,
		CheckboxGroup, Class,
		ClassLoader, Color, ColorModel,
		Compiler, Component,
		ContentHandler, DatagramPacket,
		DatagramSocket, Date, Dictionary,
		Dimension, Event, File,
		FileDescriptor,
		FilteredImageSource, FlowLayout,
		Font, FontMetrics, Graphics,
		GridBagConstraints
		GridBagLayout, GridLayout,
		Image, ImageFilter, InetAddress,
		InputStream, Insets, Math,
		MediaTracker,
		MemoryImageSource,
		MenuComponent, Number,
		Observable, OutputStream,
		PixelGrabber, Point, Polygon,
		Process, Random,
		RandomAccessFile, Rectangle,
		Runtime, SecurityManager,
		ServerSocket, Socket, SocketImpl,
		StreamTokenizer, String,
		StringBuffer, StringTokenizer,
		System, Thread, ThreadGroup,
		Thorwable, Toolkit, URL,
		URLConnection, URLEncoder,
		URLStreamHandler, Vector
Observable	java.util	Nil
Observer	java.util	Nil
OutOfMemoryError	java.lang	Nil
OutputStream	java.io	ByteArrayOutputStream,

(Contd)

(Contd)

Class	*Package*	*Subclasses*
		FileOutputStream,
		FilterOutputStream,
		PipedOutputStream,
PanelPeer	`java.awt.peer`	Nil
Panel	`java.awt`	Applet
PipedInputStream	`java.io`	Nil
PipedOutputStream	`java.io`	Nil
PixelGrabber	`java.awt.image`	Nil
Point	`java.awt`	Nil
Polygon	`java.awt`	Nil
PrintStream	`java.io`	Nil
Process	`java.lang`	Nil
Properties	`java.util`	Nil
ProtocolException	`java.net`	Nil
PushbackInputStream	`java.io`	Nil
RandomAccessFile	`java.io`	Nil
Random	`java.util`	Nil
Rectangle	`jav.awt`	Nil
RGBImageFilter	`java.awt.image`	Nil
Runnable	`java.lang`	Nil
RuntimeException	`java.lang`	ArithmeticException,
		ArrayStoreException,
		ClassCastException,
		EmptyStackException,
		IllegalArgumentException,
		IllegalMonitorStateException,
		IndexOutOfBoundsException,
		NegativeArraySizeException,
		NoSuchElementException,
		NullPointerException,
		SecurityException
Runtime	`java.lang`	Nil
ScrollbarPeer	`java.awt.peer`	Nil
Scrollbar	`java.awt`	Nil
SecurityException	`java.lang`	Nil
SecurityManager	`java.lang`	Nil
SequenceInputStream	`java.io`	Nil
ServerSocket	`java.net`	Nil
SocketException	`java.net`	Nil
SocketImplFactory	`java.net`	Nil

(Cont

(Contd)

Class	Package	Subclasses
SocketImplFactory	java.net	Nil
Socket	java.net	Nil
StackOverflowError	java.lang	Nil
Stack	java.util	Nil
StreamTokenizer	java.io	Nil
StringBufferInputStream	java.io	Nil
StringBuffer	java.lang	Nil
StringIndexOutOfBoundsException	java.lang	Nil
StringTokenizer	java.util	Nil
String	java.lang	Nil
System	java.lang	Nil
TextAreaPeer	java.awt.peer	Nil
TextArea	java.awt	Nil
TextComponentPeer	java.awt.peer	TextAreaPeer, TextFieldPeer
TextComponent	java.awt	TextArea, TextField
TextFieldPeer	java.awt.peer	Nil
TextField	java.awt	Nil
ThreadDeath	java.lang	Nil
ThreadGroup	java.lang	Nil
Thread	java.lang	Nil
Throwable	java.lang	Error, Exception
Toolkit	java.awt	Nil
UnknownError	java.lang	Nil
UnknownHostException	java.net	Nil
UnknownServiceException	java.net	Nil
UnsatisfiedLinkError	java.lang	Nil
URLConnection	java.net	Nil
URLEncoder	java.net	Nil
URLStreamHandlerFactory	java.net	Nil
URLStreamHandler	java.net	Nil
URL	java.net	Nil
UTFDataFormatException	java.io	Nil
Vector	java.util	Stack
VerifyError	java.lang	Nil
VirtualMachineError	java.lang	InternalError, OutOfMemoryError, StackOverflowError, UnknownError
WindowPeer	java.awt.peer	DialogPeer, FramePeer
Window	java.awt	Dialog, Frame

APPENDIX G

Assertion and Design by Contract

G.1 INTRODUCTION

The concept of Design By Contract (DBC) was first introduced in the Eiffel programming language. This technique specifies the interaction between various components of an application. To achieve this, it defines a contract, based on which the components of the application communicate with each other. The DBC technique uses assertions to check whether the application meets the requirements specified in the defined contract. These assertions can be used to test the assumptions made by a programmer in the Java programming language. In this appendix, we will discuss the concept of design by contract and assertion.

G.2 DESIGN BY CONTRACT

The design by contract technique allows a programmer to provide a detailed specification to create a software according to the user requirements. Based on this specification, the programmer develops the software. DBC technique uses three types of assertions to check whether the software complies with the specification. The three types of assertions are:

- **Precondition:** An application must satisfy this specified condition before calling an external component.
- **Postcondition:** An application must satisfy this specified condition after the execution of the external component.
- **Invariant:** The application must always satisfy this specified condition.

For example, consider the operation of a stack that uses precondition, postcondition, and invariant assertions. When we need to extract an element from a stack, the stack should not be empty. This condition is checked before extracting an element from a stack. This type of condition is referred to as *precondition*. When we push an element into the stack, we need to check whether the element is correctly added to the specified index. This type of condition is referred to as *postcondition*. The number of elements in the stack is greater than or equal to zero and should not exceed the capacity of the stack. This type of condition is referred to as *invariant*.

G.3 IMPLEMENTING ASSERTION

An assertion is a statement, which contains a Boolean expression that the programmer assumes to be true. If the result of the Boolean expression is true, the program execution continues. Here, the assertion ensures that the assumptions made by the programmer are correct and free from errors. If the result of the Boolean expression is false, the **AssertionError** exception will be thrown. This exception contains error information, such as file name and the line number in which the error has occurred in the program.

We use the assert statement to implement assertions in Java programs. The assert statement can be represented in two forms. One of the forms is:

```
assert Expression1;
```

Expression1 is a Boolean expression. If the result of the Boolean expression is false, the AssertionError exception is thrown without any information about the bugs that occurred in the program. Another form of the assert statement is:

```
assert Expression1 : Expression2;
```

Expression1 is a Boolean expression and Expression2 is a value, which is passed to the constructor of the **AssertionError** exception. The following code illustrates the use of the assert statement:

```
public void division()
{
double C=a/b; // b cannot be zero
}
```

In the above code, the comment line, "b cannot be equal to zero" can be replaced by using the assert statement, as shown below:

```
public void division()
{
assert b!=0;
double c=a/b;
}
```

If the expression in the assert statement is false, it specifies that the program contains errors and this process is referred to as assertion failure.

The value of the Expression2 in the second form must be a non-void expression. Consider the following example:

```
assert age>0 : "The age of a person should not be less than zero";
```

where, age>0 is a Boolean expression. The second expression in the assert statement is a String value and it is passed to the constructor of the **AssertionError** exception. If the assertion failure occurs, the string value acts as additional information about the errors that arise in the program.

Often readers may get confused with the usage of assertion and exception. The difference between the exception and assertion is:

- Exception is used to test the abnormal conditions, such as division by zero, ArrayIndexOutOf BoundsException occurred while executing the program and it does not ensure that the program is running correctly.

- Assertion is used to test the condition assumed by the programmer, and it ensures that the program is running correctly.

Compiling the Assert Statement

As we know, we use the following command to compile a Java program:

```
javac filename.java
```

The above command does not compile the programs that contain the assert statement. Therefore, we need to use the following command to compile the programs that use the assert statement:

```
java -source 1.4 filename.java
```

Enabling and Disabling Assertions

Java provides the command line parameters to enable and disable assertions. The syntax of using the command line parameter to enable assertion is:

```
-ea or -enableassertions
```

For example, the command used to run the Java file, Myfile.java, which enables assertion is:

```
java -enableassertions Myfile.java
```

Or

```
java -ea Myfile.java
```

The syntax of using the command line parameter to disable assertion is:

```
-da or disableassertions
```

For example, the command used to run the Java file, Myfile.java, which disables assertion is:

```
java -disableassertions Myfile.java
```

Or

```
java -da Myfile.java
```

G.4 ASSERTION RULES

Assertion is used to check the validity of an assumption, which is made by a programmer at the time of execution. There are certain rules that govern the usage of assertions in a program. The assertion rules are:

- Check the method arguments
- Use assertion in the default case of the Switch statement
- Make use of an assertion descriptive
- Avoid processing in an assertion condition
- Avoid catching assertion related exception
- Avoid the use of evaluating more than one condition in an assertion

Checking the Method Arguments

We need to check the values passed to the arguments of a method before performing any operation on these values. We check the argument values in the methods, public or protected, or the local package. For example, consider the following program to add two positive numbers.

```
public void addition(int a, int b)
{
  int c=a+b;
}
```

In the above code, the user may provide a negative value of the arguments that results in an inaccurate solution. Therefore, we need to check whether the values of the arguments are positive before performing the addition operation. The following code satisfies the first rule of assertion:

```
public void addition(int a, int b)
{
  if(a>0)
  {
    if(b>0)
    int c=a+b;
  }
}
```

Using Assertion in the Default Case of the Switch Statement

We can use assertions in the switch statement with no default case. In such cases, we add the default case to specify the assert statement. If none of the conditions are satisfied in the switch case and the assertion is enabled, the application causes the assertion failure and throws **AssertionError** exception.

For example, consider the following example:

```
public String Number()
{
String code;
String description;
if (number >0)
code= "POSITIVE";
IF(number<0)
code= "NEGATIVE";
switch(code)
{
case POSITIVE:
description= "The Number is Positive"
break;
case NEGATIVE:
description= "The Number is Negative";
break;
default:
assert false : "Unknown code" + code;
}
assert(description!= null) : "Provide description";
return description;
}
```

In the above code, we use the **assert** statement in the default case. This statement will be executed if the variable code does not match with any of the specified cases.

Make Use of an Assertion Descriptive

We need to provide a descriptive string message in an assert statement. It enables a programmer to understand the type of error in case the assertion failure occurs.

A program can contain a number of assert statements. If an assert statement does not contain description, the programmer cannot identify the error easily when the assertion failure occurs. For example, consider the following code segment, which contains an assert statement without description:

```
public void setName()
{
   if(name == null)
   throw new IllegalArgumentException();
}
private void setName()
{
   assert (name != null)
   ..........
}
```

In the above code, the assert statement does not contain the description. When the assertion fails, the programmer is not able to equate the error against the assertion. Therefore, the above code is rewritten as:

```
public void setName()
{
   if(name == null )
   throw new Illegal ArgumentException("name cannot be null");
}
private void setName()
{
   assert (name != null) : "name cannot be null"
   ..........
}
```

In the above code, if the assert statement fails, the message "name cannot be null" is reported to the programmer.

Avoid Processing in an Assertion Condition

When an assertion is disabled, JVM does not execute the assert statement. Therefore, the operation to be processed or performed in the assertion statement is not executed. For example, consider the following code:

```
assert list.remove("Element") : "The assertion was not in the list";
```

This is an assert statement and the assertion is disabled. Therefore, "Element" is not removed from the list. We can rewrite the above code as:

```
boolean stprocessed=list.remove("Element");
assert stprocessed : "The Element was not in the list";
```

Avoid Catching Assertion Related Exception

The act of catching exceptions related to the assertion may discard the assertion and design by contract mechanism. Some of the assertion related exceptions are **AssetError** and **IllegalArgumentException.**

Consider the following example of catching the IllegalArgumentException:

```
public void NamingInformation()
{
try
{
setName(null)
{
catch(IllegalArgumentException e)
{
}
}
public void setName(String name)
{
if(name==null)
{
throw new IllegalArgumentException("name cannot be null");
}
this.name = name;
}
```

The above code throws the IllegalArgumentException exception which arises when the name is specified as null.

Avoid Evaluating more than one Condition in an Assert Statement

We need to avoid using more than one condition in an assert statement. When we use more than one condition in an assert statement it may be difficult to find which of the conditions is not satisfied. For example, consider the following code which uses more than one condition in an assert statement:

```
public void setname(String firstName, String lastname)
{
  if((firstName==null)||(lastName==null))
  {
    throw new I11egalArgumentException("firstName or lastName cannot be
    null");
  }
}
private void setname(String firstName, String lastName)
{
  assert (firstName != null) && ( lastName != null) : "FirstName or
  LastName cannot be null";
  ..........
  ..........
}
```

In the above code, if one of the conditions fails, it is difficult for the programmer to identify which of the assumptions causes failure. Therefore, the above code can be rewritten as:

```
public void setname(String firstName,  String lastName)
{
if((firstName==null)
```

```
{
throw new IllegalArgumentException("firstName cannot be null");
}
if ((lastName=null)
{
throw new IllegalArgumentException "Last Name cannot be null");
}
}
private void setname(String firstName, String lastName)
{
assert (firstName != null ): "FirstName cannot be null";
assert(lastName != null): "Last name cannot be null";
.....
.....
}
```

G.5 CREATING A JAVA PROGRAM USING ASSERTION

In this section, we will create a simple Java program to show the usage of assertion. This program helps the reader to implement assertion. For example, consider the division of two numbers as shown in Program G.1. Here, the assert statement uses the assumption that the divisor should not be zero.

Program G.1 *Using Assertion*

```
public class division
{
   void assertcheck(int a, int b)
   {
      assert b!=0: "The value b cannot be zero";
      double c=a/b;
      System.out.println "The result is" +C);
   }
   public static void main(String args[])
   {
      division div=new division();
      div.assertcheck(5, 0);
   }
}
```

When we compile and run the above program, the **assertcheck( )** method is called and value of the rgument is passed to the parameters **a** and **b.** The value of **b** should be checked using the **assert** statement. f the assertion fails, the control will be transferred to the expression2 in the assert statement. Here, the ontrol will not be transferred to the next statement after the assert statement. The output of the above rogram is:

The value **b** cannot be zero.

APPENDIX

H

Java Version History

H.1 INTRODUCTION

Java is the most important advance in the programming technology invented during the last decade of the 20th century. Java technology is still evolving and is likely to be the primary programming language of the next millennium.

Java technology has many facets that work together to get the job done. Java environment basically consists of the following three entities:

- The language itself
- A class library (Application Program Interface (API) packages)
- A set of development tools.

Sun Microsystems, the inventors of Java, calls these three entities put together as *Java Development Kit* (JDK). This means that the JDK contains everything we need for developing Java programs. The new versions of Java are released as JDK versions by Sun Microsystems. That is, version JDK x.y means version Java x.y and vice versa.

Since the release of the original version of Java (known as Java 1.0) in May 1995, Sun Microsystems has been regularly releasing updates (changes and enhancements) of Java systems. Java 1.1 was released in March 1997 and Java 1.2 in early 1998. Fortunately, most of the changes and enhancements are related to the API packages and tools and very little changes have been introduced in the language itself. In early 1999, the Java 1.2 was renamed as Java 2 by Sun Microsystems and therefore, JDK 1.2, Java 1.2, and Java 2 all refer to the same thing.

The core API packages that contain numerous classes and interfaces have grown from around 200 in version 1.0 to more than 1500 in version 2. Table H.1 summarizes the core API packages available in Java 2 and their functions. The table shows not only the packages added during the stages 1.1 and 2 but also the number of classes and interfaces added to the various existing packages at these stages. These packages together contain more than 1500 classes and interface and define more than 13,000 methods. It is beyond the scope of this book to provide a complete description of all these classed and methods. (For more details, readers may refer to the JDK 1.2 documentation available in one of the Sun web sites or refer to the book

Java Developers Almanac 1998, Patrick Chan, Addison Wesley). More statistics of Java packages are given in Appendix I.

Table H.1 *API Packages Available in Java 2*

Sl No.	Package	Stage First Added	Total Classes and Interfaces 1.0	1.1	2	Task/Function
1.	`java.applet`	1.0	4	4	4	Provides basic functionality needed to implement applets.
2.	`java.awt`	1.0	46	61	78	Provides the standard GUI controls as well as drawing, printing and other capabilities
3.	`java.awt. accessability`	2	–	–	13	Supports the use of assistive technologies for disabled users.
4.	`java.awt.color`	2	–	–	7	Supports the ability to work with different color models.
5.	`java.awt.`	1.1	–	6	6	Supports clipboard operations, for transferring data between applications.
6.	`jawa.awt.dnd`	2	–	–	17	Supports drag–and–drop operations.
7.	`java.awt.dnd.peer`	2	–	–	3	Provides capability to access platform–dependent drag–and–drop facilities.
8.	`java.awt.event`	1.1	–	30	33	Provides foundation for processing events fired by AWT components.
9.	`java.awt.font`	2	–	–	15	Provides support for advanced font capabilities.
10.	`java.awt.geom`	2	–	–	33	Supports standard geometrical objects and transformations, in 2D.
11.	`java.awt.im`	2	–	–	3	Supports the Input Method API for internationalization.
12.	`java.awt.image`	1.0	12	14	52	A Java 2D API package that supports image processing.
13.	`java.awt.image.codec`	2	–	–	6	Supports image compression.
14.	`java.awt.image. renderable`	2	–	–	7	Supports functions for producing rendering-independent images.
15.	`java.awt.peer`	1.0	22	27	26	Provides support for interfacing with the underlying window system.
16.	`java.awt.print`	2	–	–	12	A Java 2D API package that supports printing of text and graphics.

(Contd)

Table H.1 *(Contd)*

Sl No.	Package	Stage First Added	Total Classes and Interfaces 1.0	1.1	2	Task/Function
17.	java.awt.swing	2	–	–	169	Provides foundation for the swing API, for creating completely portable GUI.
18.	java.awt.swing.border	2	–	–	10	Implements borders and border styles, around a swing component.
19.	java.awt.swing.event	2	–	–	38	Implements Swing events and event listeners.
20.	java.awt.swing.plaf	2	–	–	40	Supports the use of the pluggable look-and-feel, capabilities.
21.	java.awt.swing.plaf.basic	2	–	–	107	Provides default look-and-feel of Swing components.
22.	java.awt.swing.plaf.metal	2	–	–	42	Provides "metal" look-and-feel of Swing components.
23.	java.awt.swing.preview	2	–	–	21	Contains classes whose API have not been finalized yet.
24.	java.awt.swing.table	2	–	–	13	Implements the Swing table component for managing tables.
25.	java.awt.swing.text	2	–	–	87	Implements text processing functions such as selection, editing, etc.
26.	java.awt.swing.text.html	2	–	–	39	Provides basic HTML editing capabilities.
27.	java.awt.swing.text.rtf	2	–	–	1	Provides the capability to edit Rich Text Format documents
28.	java.awt.swing.tree	2	–	–	11	Provides capability to construct and manage tree-type data structure.
29.	java.awt.swing.undo	2	–	–	9	Provides undo and redo capabilities in applications.
30.	java.beans	1.1	–	23	25	Provides the basic Java beans functionality.
31.	java.beans.beancontext	2	–	–	19	Supports the implementation of execution environment for beans.
32	java.io	1.0	31	69	77	Performs a wide variety of input and output functions.

(Contd)

Table H.1 *(Contd)*

Sl No.	Package	Stage First Added	Total Classes and Interfaces 1.0	1.1	2	Task/Function
33.	java.lang	1.0	64	69	76	Provides support for implementing fundamental Java objects.
34.	java.lang.ref	2	–	–	6	Supports Reference Objects that are used to refer to other objects.
35.	java.lang.reflect	1.1	–	7	9	Provides capability to obtain reflective information about classes, and objects.
36.	java.math	1.1	–	2	2	Provides capability to perform arbitrary-precision arithmetic.
37.	java.net	1.0	19	26	32	Supports features for network programming.
38.	java.rmi	1.1	–	19	20	Provides capability to access objects on remote computers.
39.	java.rmi.activation	2	–	–	16	Supports persistent object reference and remote object activation.
40.	java.rmi.dgc	1.1	–	3	3	Supports functions for distributed garbage collection.
41.	java.rmi.registry	1.1	–	3	3	Supports distributed registry operations.
42.	java.rmi.server	1.1	–	23	24	Provides capabilities for supporting the server side of RMI.
43.	java.security	1.1	–	26	57	Provides basic foundation for the Security API.
44.	java.security.acl	1.1	–	8	8	Provides capability for implementing security access controls.
45.	java.security.cert	2	–	–	12	Provides support for parsing and managing digital certifications.
46.	java.security.interfaces	1.1	–	5	5	Supports implementation of the NIST digital signature of algorithm.
47.	java.security.spec	2	–	–	10	Provides specification for cryptographic keys
48.	java.sql	1.1	–	17	24	Provides support for the Java database connectivity.
49.	java.text	1.1	–	19	28	Provides support for internationalization of text and messages.

(Contd)

Table H.1 (Contd)

Sl No.	Package	Stage First Added	Total Classes and Interfaces 1.0	1.1	2	Task/Function
50.	java.util	1.0	14	26	49	Supports a variety of common programming needs.
51.	java.util.jar	2	–	–	8	Provides support for working with JAR (Java Archive) files.
52.	java.util.mime	2	–	–	3	Provides capability to work with MIME type objects.
53.	java.util.zip	1.1	–	17	17	Provides support for working with compressed files.
54.	org.omg.CORBA	2	–	–	104	Implements the foundation for supporting Java-CORBA integration.
55.	org.omg.CORBA. ContainedPackage	2	–	–	1	Describes a CORBA object in a CORBA container.
56.	org.omg.CORBA. ContainerPackage	2	–	–	1	Describes a CORBA container object.
57.	org.omg.CORBA. InterfaceDefPackage	2	–	–	1	Describes a CORBA interface definition.
58.	org.omg.CORBA. ORBPackage	2	–	–	1	Raises an exception when an invalid name is passed to an object request broker.
59.	org.omg.CORBA.	2	–	–	2	Signals exceptions related to type usage and constraints.
60.	org.omg.CORBA. portable	2	–	–	5	Supports vender-specific CORBA implementation.
61.	org.omg.CosNaming	2	–	–	22	Implements a tree-structured data type naming.
62.	org.omg.CosNaming. NamingContextPackage	2	–	–	18	Implements nodes within the tree-structured naming scheme.

H.2 CHANGES IN JAVA 1.1

As pointed out earlier, the updates include five kinds of changes:

1. Additions to the existing packages
2. Addition to new packages
3. Changes in the existing classes and members
4. Changes in the language itself
5. Changes in tools

Addition to the Existing Packages

Additions to the existing packages may include two things:

1. New classes in the existing packages
2. New members (fields, constructors, and methods) in the existing classes

As seen from Table H.1, new classes have been added to all the classes except **Applet** class. Similarly, a comparison of Tables J.2 and J.3 shows that new members have been added to many classes in almost all the existing packages. All these additions are aimed at enhancing the functionality of the existing API. Some important enhancements are listed as follows:

Abstract Windowing Toolkit (AWT) Enhancements

The additions have improved the functiona-lity of AWT to make the large-scale GUI development more feasible. Java 1.1 supports delegation-based event handling, data transfer such as cut-copy-paste, desktop color schemes, printing, mouseless operation, faster scrolling, popup menus and much more. These improvements have made Java 1.1 faster than Java 1.0.

I/O Enhancements

Java 1.1 has added character streams to the existing **java.io** package. These are like byte streams of Java 1.0 except that they operate on 16-bit Unicode characters rather than eight-bit bytes. Character streams make it easy to write programs that are independent of the user's culture and language and therefore easier to write "global programs". The process of writing a global program and ensuring that it can be used without change by anyone in the world is known as *internationalization*. In addition, two byte streams were added to support object serialization. Serialization lets us store objects and handle them with binary input/output streams.

Networking Enhancements

The 1.1 release made several enhancements to the networking package, **java.net**. It supports selected BSD-style options in the base classes and provides facility for finer granularity in reporting and handling network errors.

Native Methods Interface

Native methods are written in languages other than Java. The native methods interface from 1.0 has been completely rewritten and formalized. This interface is now known as the Java Native Interface (JNI). This provides capability for Java objects to access native methods.

Addition of New Packages

Java 1.1 has added the following new packages to provide new capabilities.

1. `java.awt.datatransfer`
2. `java.awt.event`
3. `java.beans`
4. `java.lang.reflect`
5. `java.math`
6. `java.rmi`
7. `java.rmi.dgc`
8. `java.rmi.registry`
9. `java.rmi.server`
10. `java.security`
11. `java.security.acl`
12. `java.security.interfaces`
13. `java.sql`
14. `java.text`
15. `java.util.zip`

Some major new capabilities are discussed as follows:

Security and Signed Applets

Java 1.1 supports the developments of digitally signed Java applications. It provides capability for key management, certificate management and security access controls.

Java Archive Files

Java Archive (JAR) files introduced in version 1.1 provide the capability for storing a number of files together by zipping them to shrink them, so the user can download many files at once. JAR files help us organize applets, applications, beans, and class libraries and support more efficient use of network resources.

JavaBeans Architecture

The new JavaBeans architecture provides specifications that describe Java objects suitable for reuse. The JavaBeans API allows third-party software vendors to create and ship reusable components (known as Beans), such as text, spreadsheets, graphic widgets, etc., that can be used by non-programmers to build applications.

Math Package

The new package **java.math** added to Java 1.1 contains two classes, **BigInteger** and **BigDecimal**. They provide support for performing arithmetic and bit manipulation on arbitrary precision decimal and integer numbers, without causing overflow or loss of precision.

Remote Method Invocation (RMI)

RMI API, introduced in 1.1, provides capability to create distributed Java-to-Java applications. Java objects in a local computer can invoke the methods of objects on a remote computer. The concept of object serialization is used to pass objects as parameters and return values in the remote method invocations.

Reflection

Reflection means identification of fields, constructors and methods of loaded classes and objects and using this information at runtime. These capabilities are used by Java Beans, object inspection tools, debuggers, and other Java applications and applets.

Java Database Connectivity (JDBC)

JDBC capability is provided by the package **java.sql**. This provides a uniform access to a wide range of relational databases from Java.

Changes in the Existing Classes and Methods

Many classes and methods have undergone changes from 1.0 to 1.1. Changes may be:

1. New members in the existing classes
2. Deprecation of classes
3. Deprecation of methods
4. Removal of classes
5. Removal of methods
6. Modification of design of classes
7. Modification of definition of methods

A complete description of all these changes is beyond the scope of this appendix. A brief description of classes and methods that have been deprecated or removed is given in Appendix I.

Changes in Language Itself

Changes in language itself were very minor. Bytes and shorts are accommodated as wrapped numbers by adding new classes **Byte** and **Short**. The abstract class **Number** gets two new concrete methods **byteValue** and **shortValue**.

A new class **Void** has been added as an uninstantiable place holder.

Inner Classes

One important change to the Java 1.1 is the ability to define classes as members of other classes. Such classes are called *nested classes*. Inner classes are one type of nested classes.

Instance Initializers

Java 1.0 supported initialization of only static variables (also known as class variables). Example:

```
class TestClass
{
static   {
     -------
     -------          // Initialization code
     -------
     }
}
```

Java 1.1 permits initialization of instance variables as well. Example:

```
class TestClass
{
     {
          -------
          -------     // Initialization code
          -------
     }
}
```

Array Initialization

Java 1.1 permits initialization of an array content in a **new** statement. For example, the following code creates an array of strings:

```
String [ ]    city=    new String [   ] {
                       "Madras"
                       "Delhi"
                       "Bombay" } ;
```

New Uses for Final

Java 1.1 allows us to declare the method parameters and local variables as final. However, a subclass can override a method and add or drop any final parameter modifiers. We can also deter initialization of a final variable, as long as we initialize it before it is used and assign a value to it exactly once.

H.3 CHANGES IN JAVA 2

Java 2 is a major upgrade of the core API and adds a standard extension architecture. Again the changes may be classified as follows:

1. Additions to the existing packages (Enhancements)
2. Adding new packages (New capabilities)
3. Changes in the existing class and methods
4. Changes in the language
5. Changes in tools

Enhancements in Java 2

Capabilities of java has been considerably enhanced by adding new classes to the existing packages as well as new members to almost all the existing classes. A comparison of Tables 1.3 and 1.4 (see Appendix I) will reveal this.

Security Enhancement

Java 2 provides users with the capability to specify security policies simply by editing the security permissions stored in their policy text files. Unless a permission is explicitly specified to code, it cannot access the resource that is guarded by that permission.

Java Beans Enhancement

Java 2 provides facilities to create more sophisticated JavaBeans components and applications. It provides capability to incorporate with other Beans and to learn information about their execution environment.

RMI Enhancement

Java 2 has significantly enhanced the RMI API. It supports remotely activated objects and object references that persist across multiple object activation.

JNI Enhancement

Java 2 extends Java native Interface (JNI) to incorporate new features to provide capabilities for controlling the manner in which native methods interact with the Java Virtual Machine.

JDBC Enhancement

Java 2 includes an improved version of JDBC–ODBC bridge driver and supports JDBC 2.0.

Audio Enhancement

Java 2 contains a new, high-quality sound engine that provides support for audio in applications as well as applets. The sound engine also provides support for the Musical Interface Digital Interface (MIDI) in addition to other traditional sounds.

JAR Enhancement

Java 2 JAR enhancements include improved tools for creating and updating JAR files and performing JAR I/O operations.

Reflection Enhancement

Reflection support was introduced in 1.1. Java 2 adds additional capability to identify a field, method, or constructor as suppressing Java language access controls. This facilitates the better use of reflection with the improved Java 2 security model.

New Capabilities in Java 2

In addition to improving the existing capabilities of the API, Java 2 has also added a number of new capabilities to it. Perhaps, the single most new feature is the addition of Java Foundation Classes (JFC) to Java 2. The JFC includes the functionalities such as Swing, Java 2D, Drag-and-Drop and Accessibility. Besides integrating JFC, Java 2 provides a number of other new capabilities. The new packages added include the following:

```
 1. java.awt.accessibility
 2. java.awt.color
 3. java.awt.dnd
 4. java.awt.dnd.peer
 5. java.awt.font
 6. java.awt.geom
 7. java.awt.im
 8. java.awt.image.codec
 9. java.awt.image.renderable
10. java.awt.print
11. java.awt.swing
12. java.awt.swing.border
13. java.awt.swing.event
14. java.awt.swing.plaf
15. java.awt.swing.plaf.basic
16. java.awt.swing.plaf.metal
17. java.awt.swing.preview
18. java.awt.swing.table
19. java.awt.swing.text
20. java.awt.swing.text.html
21. java.awt.swing.text.rtf
22. java.awt.swing.tree
23. java.awt.swing.undo
24. java.beans.beancontext
25. java.lang.ref
26. java.rmi.activation
27. java.security.cert
28. java.security.spec
29. java.util.jar
30. java.util.mime
31. org.omg.CORBA
32. org.omg.CORBA.ContainedPackage
33. org.omg.CORBA.ContainerPackage
34. org.omg.CORBA.InterfaceDefPackage
35. org.omg.CORBA.ORBPackage
36. org.omg.CORBA.TypeCodePackage
37. org.omg.CORBA.portable
38. org.omg.CosNaming
39. org.omg.CosNaming.NamingContextPackage
```

Swing

Swing is the code word used by the JavaSoft team for the improved AWT. Swing implements a new set of GUI components with a "pluggable" look and feel. Swing is implemented completely in Java. Pluggable look and feel architecture allows us to design a single set of GUI components that can automatically have the look and feel of any OS platform.

Java 2D

The new Java 2D API includes a set of tools for dealing with two-dimensional drawings and images. These include provision for colorspaces, text, line art and printing.

Accessibility

Accessibility API provides support for the use of *assistive technologies*, such as screen magnifiers, speech recognition systems, and Braile terminals intended for use by disabled users.

Drag and Drop

Drag and Drop capability of Java 2 facilitates data transfer across Java and native applications, between Java applications, and within a single application.

Java IDL

Java IDL in Java 2 provides a set of tools for interfacing Java objects with CORBA (Common Object Request Broker Architecture) objects and for developing CORBA objects in Java. It also includes a Java ORB (Object Request Broker) and an ORB name server.

Collections

The Collections API provides an implementation-independent framework for working with collection of objects such as sets, maps, lists, and linked lists.

Package Version Identification

A new capability of Java 2 allows applets and applications to obtain version information about a particular Java package at runtime.

Reference Objects

Reference objects (introduced in version 2) stores references to other objects. This feature can be used to implement object-catching mechanisms.

Input Method API

The new Input Method API provides support for Java's internationalisation. This enables all text-editing components to receive foreign language text input through input methods. It currently supports Japanese, Chinese and Korean languages.

Language Changes

There are not major changes in language. Only three methods of **Thread** class, **stop( ), suspend( ),** and **resume( )** have been depreciated because of errors and inconsistencies caused by them. Instead of using the **stop( )** method, it is proposed that a thread may monitor the state of a shared variable and stop execution by returning from its **run( )** method. Similarly, a thread may suspend and resume its execution based on the value of shares variables (by monitoring interface events).

Tools Changes

Java 2 has improved the tools available in the earlier versions and also added new tools. The **javakey** tool of 1.1 has been replaced by the new **keytool** and **javasinger** tools. The Java 2 now includes the following tools:

- **keytool** for maintaining a database of key pairs and digital certificates.
- **javasinger** for signing JAR files and verifying the signatures of signed files.
- **policytool** for creating and modifying the files that define security policy.
- **tnameserv** for implementing CORBA Common Object Services (COS) Naming Service.
- **rmid** for remote activation system daemon.

H.4 PERFORMANCE ENHANCEMENTS

The enhancements and new features added to Java 2 has considerably improved the performance of Java which has been a subject of criticism. Given below are some of the performance improvements achieved in Java 2.

- Improvements in multithreading performance
- Reduction in memory usage for string constants
- Faster memory allocation and garbage collection
- Improvements in the performance of the thread monitor methods

- Support of native libraries for some critical API classes
- Inclusion of just-in-time (JIT) compilers with Java 2

H.5 WHAT'S NEW IN JAVA 6

Java SE 6 is the latest version of Java that was released on December 11, 2006. It was codenamed as Mustang. Java 6 is more efficient and robust in comparison to the previous versions of Java. Higher efficiency is induced into this version through several key additions and updates. These additions have provided greater flexibility and newer options for the Java developers. Some of the key features of Java SE 6 are:

- It supports blending of scripting languages with java code.
- It provides extensive support for all JDBC databases, thus eliminating the need for configuring databases. As a result, the development of applications, which require fetching data from the database, has become simpler.
- Java SE 6 comes bundled with several new options for developing GUI-based applications. The major add-on is **SwingWorker**, which facilitates threading in the GUI applications.
- It offers specialized tools for monitoring and managing memory-heaps and other system related information.
- It offers several new features on the security front such as cryptographic services, Public Key Infrastructure (PKI) and XML-Digital signature.

Table H.2 lists some of the key additions in Java SE 6:

Table H.2 *Additions in Java SE 6*

Interfaces	Deque BlockingDeque NavigableSet NavigableMap ConcurrentNavigableMap isObjectMonitorUsageSupported isSynchronizerUsageSupported findDeadlockedThreads
Classes	ArrayDeque ConcurrentSkipListSet ConcurrentSkipListMap LinkedBlockingDeque AbstractMap.SimpleEntry AbstractMap.SimpleImmutableEntry Console
Methods	getTotalSpace getFreeSpace getUsableSpace setWritable setReadable setExecutable

APPENDIX I

Deprecated Classes and Methods

I.1 INTRODUCTION

As a part of the effort to enhance the performance and capabilities of the Java language, Sun Microsystems has altered and eliminated many classes and methods during upgradations. The altered methods have been added as new methods and the older ones have been retained in order to maintain backward compatibility with older versions of Java. However, the older methods have been marked "deprecated".

A deprecated method means that it has lost its importance and likely to be phased out of future Java versions. Although programs that use deprecated methods will still compile and work, the compiler will generate warning messages. It is recommended that programs be modified to eliminate the use of any deprecated methods and classes due to two reasons.

1. Modified programs will retain compatibility with future releases of Java.
2. Many new methods provide better implementations and therefore make programs faster and more efficient.

I.2 DEPRECATED CLASSES AND METHODS OF VERSION 1.0

A large number of classes and methods of version 1.0 have been declared deprecated. This appendix gives package-wise tables (Tables I.1 to I.4) that list class-wise methods that have been declared deprecated in Java 1.1. Tables also indicate alternative replacements. Table I.5 gives a list of version 1.0 classes that have been declared deprecated totally.

Table I.1 *List of Deprecated 1.0 Methods in Java.lang*

Classes	*Methods*	*Replacement*
Character	isJavaLetter (char)	isJavaIdentifierStart (char)
	isJavaLetterOrDigit (char)	isJavaIdentifierPart (char)
	isSpace (char)	isWhitespace (char)
ClassLoader	defineClass (byte [ ], int, int)	defineClass (byte [ ], int,int)
	toLocaleString ( )	Use java.text. Format and its subclasses
	toGMTString ( )	
	parse (String s)	
Runtime	getLocalizedInputStream (InputStream in)	BufferedReader (InputStream)
		InputStreamReader (InputStream)
	getLocalizedOutputStream (OutputStream out)	BufferedWriter (OutputStream)
		OutputStreamWriter(OutputStream)
		PrintWriter (OutputStream)
String	String (byte [ ], int, int, int)	String (byte [ ], String)
	String (byte [ ], int)	String(byte[ ],int,int,String)
	getBytes (int, int, byte [ ], int)	byte [ ] setbytes ( )
System	getenv (String)	String get property (String)

Table I.2 *List of Deprecated 1.0 Methods in Java.util*

Classes	*Methods*	*Replacement*
Date Date	(int, int, int)	Create a java.util.GregorianCalendar object and use its getTime method to convert it to a Date
	Date (int, int, int, int, int)	
	Date (int, int, int, int, int, int)	
	Date (String)	
	UTC (int, int, int, int, int, int)	Create a java.util.GregorianCalendar object and use its setters and getters.
	getTimezoneOffset ( )	
	getYear ( )	
	setYear (int)	
	getMonth ( )	
	setrMonth (int)	
	getDate ( )	

(Contd)

Table I.2 (*Contd*)

Classes	*Methods*	*Replacement*
	setDate (int)	
	getDay ( )	
	getHours ( )	
	setHours (int)	
	getMinutes ( )	
	setMinutes (int)	
	getSeconds ( )	
	setSeconds (int)	
	toLocaleString ( )	Use java.txt.Format and its classes
	toGMTString ( )	
	parse (String s)	

Table I.3 *List of Deprecated 1.0 Methods in Java.io*

Classes	*Methods*	*Replacement*
ByteArrayOutputStream	toString (int)	toString (String)
DataInputStream	String readLine ( )	BufferedReader.readLine()
StreamTokenizer	StreamTokenizer (InputStream)	StreamTokenizer (Reader)

Table I.4 *List of Deprecated 1.0 Methods in Java.awt*

Classes	*Methods*	*Replacement*
BorderLayout	addLayout Component (String, Component)	addLayoutComponent (Component, Object)
CardLayout	addLayoutComponent (String, Component)	addLayoutComponent (Component, Object)
CheckboxGroup	getCurrent ( )	getSelectedCheckbox ( )
	setCurrent (checkbox)	setSelectedCheckbox (Checkbox)
Choice	countItems ( )	getItemCount ( )
Component	getPeer ( )	None
	enable ( )	setEnabled (boolean)
	enable (boolean)	setEnabled (boolean)
	disable ( )	setEnabled (boolean)
	show ( )	setVisible (boolean)
	show (boolean)	setVisible (boolean)
	hide ( )	setVisible (boolean)
	location ( )	getLocation ( )
	move (int, int)	setLocation (int, int)

(*Contd*)

Table I.4 (*Contd*)

Classes	*Methods*	*Replacement*
	size ( )	getSize ( )
	resize (int,int)	setSize (int, int)
	resize (Dimension)	setSize (Dimension)
	bounds ( )	getBounds ( )
	reshape (int, int, int, int)	setBounds (int, int, int, int,)
	preferredSize ( )	getPreferredSize ( )
	minimumSize ( )	getMinimumSize ( )
	layout ( )	doLayout ( )
	inside (int, int)	contains (int, int)
	locate (int, int)	getComponentAt (int, int)
	deliverEvent (Event)	dispatchEvent (AWTEvent)
	postEvent (Event)	dispatchEvent (AWTEvent)
	handleEvent (Event)	processEvent (AWTEvent)
	mouseDown(Event,int,int)	processMouseEvent (MouseEvent)
	mouseDrag(Event,int,int)	processMouseMotion Event (MouseEvent)
	mouseUp(Event,int,int)	processMouseEvent (MouseEvent)
	mouseMove(Event,int,int)	processMouseMotionEvent (MouseEvent)
	mouseEnter(Event,int,int)	processMouseEvent (MouseEvent)
	mouseExit(Event,int,int)	processMouseEvent (MouseEvent)
	keyDown (Event, int)	processKeyEvent (KeyEvent)
	keyUp (Event, int)	processKeyEvent (KeyEvent)
	action (Event, Object)	Should register this component as Action Listener on component which fires action events
	lostFocus(Event,Object)	processFocusEvent (FocusEvent)
	nextFocus ( )	transferFocus ( )
Container	countComponents ( )	getComponentCount ( )
	insets ( )	getInsets ( )
	layout ( )	doLayout ( )
	preferredSize ( )	getPreferredSize ( )
	minimumSize ( )	getMinimumSize ( )
	deliverEvent (Event)	dispatchEvent (AWTEvent e)
	locate (int, int)	getComponentAt (int, int)
	nextFocus (Component)	transferFocus (Component)
Frame	setCursor (int)	Component.setCursor (Cursor)
	getCursorType ( )	Component.setCursor (Cursor)
		Component.getCursor ( )
Graphics	getClipRect ( )	getClipBounds ( )

(*Contd*)

Table 1.4 (*Contd*)

Classes	*Methods*	*Replacement*
List	countItems ( )	getItemCount ( )
	clear ( )	removeAll ( )
	isSelected (int)	isIndexSelected (int)
	allowsMultipleSelections ( )	isMultipleMode ( )
	setMultipleSelections (boolean)	setMultipleMode (boolean)
	preferredSize (int)	getPreferredSize (int)
	preferredSize ( )	getPreferredSize ( )
	minimumSize (int)	getMinimumSize (int)
	minimumSize ( )	getMinimumSize ( )
	delItems (int, int)	Not for public use in the future. This method is expected to be retained only as a package private method.
Menu	countItems ( )	getItemCount ( )
	countMenus ( )	getMenuCount ( )
MenuBar	getPeer ( )	None
MenuComponent	enable ( )	setEnabled (boolean)
MenuItem	enable (boolean)	setEnabled (boolean)
	disable ( )	setEnabled (boolean)
Polygon	getBoundingBox ( )	getBounds ( )
	inside (int, int)	contains (int, int)
Rectangle	reshape(int,int,int,int)	setBounds (int, int, int, int)
	move (int, int)	setLocation (int, int)
	reSize (int, int)	setSize (int, int)
	inside (int, int)	contains (int, int)
Scrollbar	getVisible ( )	getVisibleAmount ( )
	setLineIncrement (int v)	setUnitIncrement (int)
	getLineIncrement ( )	getUnitIncrement ( )
	setPageIncrement (int v)	setBlockIncrement (int)
	getPageIncrement ( )	getBlockIncrement ( )
ScrollPane	layout ( )	doLayout ( )
TextArea	insertText (String, int)	insert (String, int)
	appendText (String)	append (String)
	replaceText(String,int,int)	replaceRange (String, int, int)
	preferredSize (int, int)	getPreferredSize (int, int)
	preferredSize ( )	getPreferredSize ( )
	minimumSize (int, int)	getMinimumSize (int, int)
	minimumSize ( )	getMinimumSize ( )

(Conti

Table I.4 (*Contd*)

Classes	*Methods*	*Replacement*
TextField	setEchoCharacter (char)	SetEchochar (char)
	preferredSize (int)	GetPreferredSize (int)
	preferredSize ( )	getPreferredSize ( )
	minimumSize (int)	GetMinimumSize (int)
	minimumSize ( )	GetMinimumSize ( )
Window	nextFocus (Component)	TransferFocus (Component)
	postEvent (Event)	DispatchEvent (AWTEvent)

Table I.5 *List of Deprecated 1.0 Methods in Java.net*

Classes	*Methods*	*Replacement*
Socket	Socket (String, int, boolean)	Use DatagramSocket's constructors
	Socket (inetAddress, int, boolean)	Use DatagramSocket's constructors

Table I.6 *List of Deprecated Classes*

Class	*Package*
EventListener	java.util
EventObject	java.util
LineNumberInputStream	java.io
PrintStream	java.io
Serializable	java.io
StringBufferInputStream	java.io

I.3 DEPRECATED CLASSES AND MEMBERS OF VERSION 1.1

Some of the classes and members of version 1.1 have been removed or declared deprecated in version 2. They are summarized in Tables I.7 and I.8.

Table I.7 *Deprecated Members of Version 1.1*

Package	*Class*	*Members*
java.awt	Frame	CROSSHAIR_CURSOR
		DEFAULT_CURSOR
		E_RESIZE_CURSOR
		HAND_CURSOR

(*Contd*)

Table 1.7 (*Contd*)

Package	*Class*	*Members*
		MOVE_CURSOR
		N_RESIZE_CURSOR
		NE_RESIZE_CURSOR
		NW_RESIZE_CURSOR
		S_RESIZE CURSOR
		SE_RESIZE CURSOR
		SW_RESIZE CURSOR
		TEXT_CURSOR
		W_RESIZE_CURSOR
		WAIT_CURSOR
java.awt.image	BufferedImage	getGraphics ( )
java.awt	List	addItem ( )
		delItem ( )
java.awt.swing	AbstractButton	getLabel ( )
		setLabel ( )
java.io	ObjectInputStream	readLine ( )
java.lang	SecurityManager	classDepth ( )
		classLoaderDepth ( )
		getInCheck ( )
		inCheck ( )
		inClass ( )
		inClassLoader ( )
	Thread	resume ( )
		stop ( )
		suspend ( )
	ThreadGroup	resume ( )
		stop ( )
		suspend ( )
java.rmi.registry	RegistryHandler	registryImpl ( )
		registryStub ( )
java.security	Certificate	Entire class
	Identity	addCertificate ( )
		certificates ( )
		removeCertificate ( )
	Signature	getParameter ( )
		setParameter ( )
	SignatureSpi	engineGetParameter ( )
		engineSetParameter ( )

(*Contd*)

Table I.7 (*Contd*)

Package	*Class*	*Members*
java.sql	Date	Date ( )
		getHours ( )
		getMinutes ( )
		getSeconds ( )
		setHours ( )
		setMinutes ( )
		setSeconds ( )
	DriverManager	getLogStream ( )
		setLogStream ( )
	PreparedStatement	setUnicodeStread ( )
	ResultSet	getUnicodeStream ( )
	Time	getDate ( )
		getDay ( )
		getMonth ( )
		getYear ( )
		setDate ( )
		setMonth ( )
		setYear ( )
	TimeStamp	TimeStamp ( )

Table I.8 *Removed Members of Version 1.1*

Package	*Class*	*Members*
java.awt.peer	ActiveEvent	dispatch ( )
java.net	URLConnection	fileNameMap
java.rmi	RMISecurityManager	checkAwtEventQueueAccess ( )
		checkCreateClassLoader ( )
		checkDelete ( )
		checkExec ( )
		checkExit ( )
		checkLink ( )
		checkListen ( )
		checkMemberAccess ( )
		checkMulticast ( )
		checkPrintJobAccess ( )
		checkPropertiesAccess ( )
		checkPropertyAccess ( )
		checkRead ( )

(*Contd*)

Table I.8 (*Contd*)

Package	*Class*	*Members*
		checkSecurityAccess ( )
		checkSetFactory ( )
		checkSystemClipboardAccess ( )
		checkTopLevelWindow ( )
		checkWrite ( )
java.security	KeyPairGenerator	generateKeyPair ( )
		initialize ( )
	MessageDigest	engineDigest ( )
		engineReset ( )
		engineUpdate ( )
	Signature	enginegetParameter ( )
		engineInitSign ( )
		engineInitVerify ( )
		engineSetParameter ( )
		engineSign ( )
		engineUpdate ( )
		engineVerify ( )
java.util	GregorianCalendar	after ( )
		before ( )
		equals ( )
	Vector	toString ( )

Since Java classes undergo continuous revision and modifications, the list of removed/deprecated methods and classes given here may not be entirely complete. It is only intended as a quick reference. For a latest list; the reader should consult the JDK documentation available in one of the Sun web sites.

We can obtain a complete list of deprecated methods in a program by using the **javac** compiler with the *–deprecation* option, when the compiler generates warning messages about usage of such methods.

APPENDIX J

Statistics of Java Packages

Table J.1 *Growth of Java from 1.0 to 2*

Java Version	Packages	Class and Interfaces			Members			
		Classes	Interfaces	Total	Fields	Constructor	Methods	Total
1.0	8	172	40	212	261	319	1545	2125
1.1	23	391	113	504	926	701	3851	5478
2	62	1287	305	1592	3107	2095	13635	18837

Table J.2 *Contents of Java 1.0 Packages*

Package	Classes	Interfaces	Total Members
`java.applet`	1	3	38
`java.awt`	44	2	740
`java.awt.image`	9	3	124
`java.awt.peer`	0	22	84
`java.io`	28	3	340
`java.lang`	62	2	505
`java.net`	16	3	148
`java.util`	12	2	146

Table J.3 *Contents of Java 1.1 Packages*

Package	Classes	Interfaces	Total Members
`java.applet`	1	3	39
`java.awt`	54	7	1317
`java.awt.datatransfer`	4	2	29

(Contd)

Table J.3 (*Contd*)

Package	Classes	Interfaces	Total Members
java.awt.event	19	11	304
java.awt.image	11	3	155
java.awt.peer	0	27	117
java.beans	17	6	147
java.io	61	8	645
java.lang	67	2	660
java.lang.reflect	6	1	98
java.math	2	0	86
java.net	22	4	254
java.rmi	18	1	68
java.rmi.dgc	2	1	10
java.rmi.registry	1	2	13
java.rmi.server	16	7	96
java.security	21	5	158
java.security.acl	3	5	30
java.security.interfaces	0	5	8
java.sql	9	8	423
java.text	18	1	315
java.util	23	3	355
java.util.zip	16	1	151

Table J.4 *Contents of Java 2 Packages*

Package	Classes	Interfaces	Total Members
java.applet	1	3	40
java.awt	64	14	1640
java.awt.accessibility	6	7	193
java.awt.color	7	0	179
java.awt.datatransfer	4	2	45
java.awt.dnd	13	4	134
java.awt.dnd.peer	0	3	19
java.awt.event	20	13	356
java.wt.font	13	2	353
java.awt.geom.	32	1	613
java.awt.im	2	1	27
java.awt.image	42	10	794
java.awt.image.codec	6	0	109
java.awt.image.renderable	4	3	94
java.awt.peer	0	26	116
java.awt.print	9	3	176
java.awt.swing	147	22	3256
java.awt.swing.border	9	1	131
java.awt.swing.event	19	19	167
java.awt.swing.plaf	39	1	135
java.awt.swing.plaf.basic	107	0	1788
java.awt.swing.plaf.metal	42	0	446
java.awt.swing.preview	20	1	257
java.awt.swing.table	9	4	266
java.awt.swing.text	66	21	961

(*Contd*)

Table J.4 (*Contd*)

Package	*Classes*	*Interfaces*	*Total Members*
java.awt.swing.text.html	39	0	100
java.awt.swing.text.rtf	1	0	7
java.awt.swing.tree	4	7	202
java.awt.swing.undo	7	2	97
java.beans	17	8	178
java.beans.beancontext	11	8	175
java.io	66	11	733
java.lang	72	4	802
java.lang.ref	6	0	21
java.lang.reflect	8	1	104
java.math	2	0	88
java.net	28	4	298
java.rmi	19	1	53
java.rmi.activation	12	4	68
java.rmi.dgc	2	1	10
java.rmi.registry	1	2	14
java.rmi.server	17	7	119
java.security	51	6	342
java.security.acl	3	5	30
java.security.cert	11	1	80
java.security.interfaces	0	5	8
java.security.spec	8	2	27
java.sql	10	18	649
java.text	20	4	352
java.util	38	13	717
java.util.jar	8	0	75
java.util.mime	3	0	29
java.util.zip	16	1	157
org.omg.CORBA	78	26	661
org.omg.CORBA.ContainedPackage	1	0	4
org.omg.CORBA.ContainerPackage	1	0	5
org.omg.CORBA.InterfaceDefPackage	1	0	10
org.omg.CORBA.ORBPackage	1	0	1
org.omg.CORBA.TypeCodePackage	2	0	2
org.omg.CORBA.portable	4	1	95
or.omg.CosNaming	20	2	138
org.omg.CosNaming. NamingContextPackage	18	0	91

APPENDIX

K

SCJP Exam Model Questions

This appendix provides a number of true/false, multiple-choice, and short-answer questions which are modeled on the Sun Certified Java Programmer (SCJP) examination conducted by Sun Microsystems, the creator of Java language. Readers must try to analyze and understand each question before attempting to answer it. They are also encouraged to verify the answers by actually testing the code snippets given. This would enable them not only to check their answers but also to enhance their understanding of Java concepts considerably.

Part A: True/False Questions

1. The name of a Java program file must match the name of the class with the extension .java
   A. True B. False
2. Two methods cannot have the same name in Java.
   A. True B. False
3. The modulus operator (%) can be used only with integer operands.
   A. True B. False
4. Declarations can appear anywhere in the body of a Java method.
   A. True B. False
5. All the bitwise operators have the same level of precedence in java.
   A. True B. False
6. When x is a positive number, the operations x > > 2 and x > > > 2 both produce the same result.
   A. True B. False
7. If a = 10 and b = 15, then the statement x = (a > b)? a : b;
   assigns the value 15 to x.
   A. True B. False
8. In evaluating a logical expression of type
   ```
   boolean expression1 && boolean expression2
   ```
   both the boolean expressions are not always evaluated.
   A. True B. False
9. In evaluating the expression (x = = y && a < b) the boolean expression x = = y is evaluated first and then a < b is evaluated.
   A. True B. False

10. The **default** case is always required in the switch selection structure.
    A. True B. False
11. The **break** statement is required in the default case of a switch selection structure.
    A. True B. False
12. The expression (x = = y && a < b) is true if ether x = = y is true or a < b is true.
    A. True B. False
13. A variable declared inside the **for** loop control cannot be referenced outside the loop.
    A. True B. False
14. Java always provides a default constructor to a class.
    A. True B. False
15. When present, **package** must be the first noncomment statement in the file.
    A. True B. False
16. The **import** statement is always the first noncomment statement in a Java program file.
    A. True B. False
17. Objects are passed to a method by use of call-by-reference.
    A. True B. False
18. It is perfectly legal to refer to any instance variable inside of a **static** method.
    A. True B. False
19. When we implement an interface method, it should be declared as **public**.
    A. True B. False
20. We can overload methods with differences only in their return type.
    A. True B. False
21. It is an error to have a method with the same signature in both the super class and its subclass.
    A. True B. False
22. A constructor must always invoke its super class constructor in its first statement.
    A. True B. False
23. Subclasses of an abstract class that do not provide an implementation of an abstract method, are also abstract.
    A. True B. False
24. Any class may be inherited by another class in the same package.
    A. True B. False
25. Any method in a super class can be overridden in its subclass.
    A. True B. False
26. One of the features of Java is that an array can store many different types of values.
    A. True B. False
27. An individual array element that is passed to a method and modified in that method will contain the modified value when the called method completes execution.
    A. True B. False
28. Members of a class specified as a **private** are accessible only to the methods of the class.
    A. True B. False
29. A method declared as **static** cannot access **non-static** class members.
    A. True B. False
30. A **static** class method can be invoked by simply using the name of the method alone.
    A. True B. False
31. It is an error, if a class with one or more **abstract** methods is not explicitly declared **abstract**.
    A. True B. False

32. It is perfectly legal to assign an object of a super class to a subclass reference without a cost.
A. True B. False
33. It is perfectly legal to assign a subclass object to a super class reference.
A. True B. False
34. Every method of a **final** class is implicitly **final**.
A. True B. False
35. All methods in an **abstract** class must be declared **abstract**.
A. True B. False
36. When the **String** objects are compared with = =, the result is true if the strings contain the same values.
A. True B. False
37. A **String** object cannot be modified after it is created.
A. True B. False
38. The length of a **String** object **s1** can be obtained using the expression **s1.length**.
A. True B. False
39. A **catch** can have comma-separated multiple arguments.
A. True B. False
40. It is an error to catch the same type of exception in two different **catch** blocks associated with a particular **try** block.
A. True B. False
41. Throwing an **Exception** always causes program termination.
A. True B. False
42. Every call to **wait** has a corresponding call to **notify** that will eventually end the waiting.
A. True B. False
43. Declaring a method **synchronized** guarantees that the deadlock cannot occur.
A. True B. False
44. The programmer must explicitly create the **System.in** and **System.out** objects.
A. True B. False
45. If the file-position pointer points to a location in a sequential file other than the beginning, we must use the **seek** method to bring the pointer to the beginning, to read from the beginning of the file again.
A. True B. False
46. To delete a file, we can use an instance of class **File**.
A. True B. False
47. A panel cannot be added to another panel.
A. True B. False
48. Frames and applets cannot be used together in the same program.
A. True B. False
49. A **final** class may not have any **abstract** methods.
A. True B. False
50. A class may be both **abstract** and **final**.
A. True B. False
51. A thread wants to make a second thread ineligible for execution. To achieve this, the first thread can call the **yield( )** method on the second thread.
A. True B. False
52. A thread can make a second thread ineligible for execution by calling the **suspend( )** method on the second thread.

A. True B. False

53. A Java monitor must either extend **Thread** class or implement **Runnable** interface.
A. True B. False

54. The **CheckboxGroup** class is a subclass of the **Component** class.
A. True B. False

55. If a frame uses a Grid layout manager and does not contain any panels, then all the components within the frame are of the same width and height.
A. True B. False

56. With a Border layout manager, the component at the centre gets all the space that is left over, after the components at North and South have been considered.
A. True B. False

57. The CODE value in an <APPLET> tag must name a class file that is in the same directory as the calling HTML page.
A. True B. False

58. If **getParameter( )** returns null, then assigning the return value to a variable of type **String** may cause an exception to be thrown.
A. True B. False

59. It is possible to use the **File** class to list the contents of the current working directory.
A. True B. False

60. **Reader** class has a method that can read and return floats and doubles.
A. True B. False

Part B: Multiple-choice Questions

1. The range of values for the long type data is
A. -2^{31} to $2^{31} - 1$ B. -2^{64} to 2^{64}
C. -2^{63} to $2^{63} - 1$ D. -2^{32} to $2^{32} - 1$

2. Which of the following represent(s) of a hexadecimal number?
A. 570 B. (hex) 5
C. 0X9F D. 0X5

3. Which of the following assignments are valid?
A. float x = 123.4; B. long m = 023;
C. int n = (int)false; D. double y = 0X756;

4. The default value of char type variable is
A. '\u0020' B. '\u00ff
C. " " D. '\u0000'

5. What will be the result of the expression 13 & 25?
A. 38 B. 25
C. 9 D. 12

6. What will be result of the expression 9 | 9?
A. 1 B. 18
C. 9 D. None of the above

7. Which of the following are correct?
A. int a = 16, a >> 2 = 4
B. int b = –8, b >> 1 = –4
C. int a = 16, a >>> 2 = 4

D. int b = –8, b >>> 1 = – 4
E. All the above

8. What will be the values of x, m, and n after execution of the following statements?

```
int x, m, n;
m = 10;
n = 15;
x = ++m + n++;
```

A. x = 25, m = 10, n = 15 B. x = 27, m = 10, n = 15
C. x = 26, m = 11, n = 16 D. x = 27, m = 11, n = 16

9. If m and n are int type variables, what will be the result of the expression

```
m % n
```

when m = 5 and n = 2?
A. 0 B. 1
C. 2 D. None of the above

10. If m and n are int type variables, what will be the result of the expression

```
m % n
```

when m = – 14 and n = –3?
A. 4 B. 2 C. –2
D. –4 E. None of the above

11. Consider the following statements:

```
int x = 10, y = 15;
x = ( (x<y) ? (y+x) : (y-x) ;
```

What will be the value of x after executing these statements?
A. 10 B. 25 C. 15
D. 5 E. Error. Cannot be executed.

12. Which of the following operators are overloaded for String objects?
A. – B. + C. + =
D. & E. << F. None of these

13. What is the result of the expression

```
(1 & 2) + (3 | 4)
```

in base ten.
A. 1 B. 2 C. 8
D. 7 E. 3

14. Which of the following will produce a value of 22 if x = 22.9?
A. ceil(x) B. round(x) C. rint(x)
D. abs(x) E. floor(x)

15. Which of the following will produce a value of 10 if x = 9.7?
A. floor(x) B. abs(x) C. rint(x)
D. round(x) E. ceil(x)

16. Which of the following expressions are illegal?
A. (10 | 5) B. (false && true) C. boolean x = (boolean)10;
D. float y = 12.34;

17. Which of the following lines will not compile?
    1. byte b1 = 5, b2 = 3, b3;
    2. short s = 25;
    3. b2 = s;
    4. b3 = b1 * b2;

    A. Line 1 only B. Line 3 only C. Line 4 only
    D. Line 1 and Line 4 only E. Line 3 and Line 4 only
18. Which of the following are illegal loop constructs?
    A. while(int i > 0)
    {i- -; other statements;}
    B. for(int i = 10, int j = 0; i+j > 5; i = i−2, j++)
    {
    Body statements
    }
    C. int i = 10;
    while (i)
    {
    Body statements
    }
    D. int i = 1, sum = 0;
    do {loop statements}
    while(sum < 10 || i<5);
19. Consider the following code

```
if (number >= 0)
    if (number > 0)
        System.out.println ("Number is positive") ;
    else
        System.out.println ("Number is negative") ;
```

    What will be the output if number is equal to 0?
    A. Number is negative B. Number is positive
    C. Both A and B D. None of the above
20. Which of the following control expressions are valid for an if statement?
    A. an integer expression B. a boolean expression
    C. either A or B D. Neither A nor B
21. In the following code snippet, which lines of code contain error?
    1. int j = 0;
    2. while(j < 10) {
    3. j++;
    4. if(j = = 5) continue loop;
    5. System.out.println("j is" + j); }

    A. Line 2 B. Line 3 C. Line 4
    D. Line 5 E. None of the above
22. Consider the following code:

```
char c = 'a' ;
switch (c)
{
```

```
        case 'a' :
        System.out.println ("A") ;
        case 'b' :
        System.out.println ("B") ;
        default:
        System.out.println ("C") ;
    }
```

For this code, which of the following statement is true?

A. output will be A
B. output will be A followed by B
C. output will be A, followed by B, and then followed by C
D. code is illegal and therefore will not compile

23. Consider the following class definition.

```
class Student extends String
{
}
```

What happens when we try to compile this class?

A. Will not compile because class body is not defined
B. Will not compile because the class is not declared **public**
C. Will not compile because String is **abstract**
D. Will not compile because String is **final**
E. Will compile successfully.

24. What is wrong in the following class definitions?

```
abstract class Print
{
    abstract show ( ) ;
}
class Display extends Print
{
}
```

A. Nothing is wrong
B. Wrong. Method **show( )** should have a return type
C. Wrong. Method **show( )** is not implemented in Display
D. Wrong. **Display** does not contain any members

25. What is the error in the following class definition?

```
abstract class XY
{
    abstract sum (int x, int y) { }
}
```

A. Class header is not defined properly
B. Constructor is not defined
C. Method is not defined properly
D. No error

26. Consider the following class definitions:

```
class maths
{
```

```
        Student student1;
}
class Student
{
        String name;
}
```

This code represents:

A. an 'is a' relationship B. a 'has a' relationship
C. both D. neither

27. Consider the following class definition:

```
class A extends B
{
        public A (int x) { }
        public A (int x, int y)
        {
                super (x, y) ;
        }
}
```

Which of the following are legal statements to construct A type objects?

A. A a = new A( );
B. A a = new A(4, 2, 7);
C. A a = new A(5, 6);
D. A a = new A(10);
E. A a = new A(Base(4, 5), 6);

28. Which of the following are overloading the method

```
int sum(int x, int y) { }
```

A. int sum(int x, int y, int z) { }
B. float sum(int x, int y) { }
C. int sum(float x, float y) { }
D. int sum(int a, int b) { }
E. float sum(int x, int y, float z) { }

29. What is the error in the following code?

```
class Test
{
        abstract void display ( ) ;
}
```

A. No error
B. Method **display( )** should be declared as **static**
C. **Test** class should be declared as **abstract**
D. **Test** class should be declared as **public**

30. Which of the following statements are true?

1. We cannot use abstract classes to instantiate objects directly.
2. The abstract methods of an abstract class must be defined in its subclass.
3. We cannot declare abstract constructors.
4. We may declare abstract static methods.

A. Line 1 only
B. Line 2 only
C. Line 1 and line 2 only
D. Line 1, line 2 and line 3 only
E. All are true

31. Which keyword can protect a class in a package from accessibility by the classes outside the package?
A. private
B. protected
C. final
D. don't use any keyword at all (make it default)

32. We would like to make a member of a class visible in all subclasses regardless of what package they are in. Which one of the following keywords would achieve this?
A. private
B. protected
C. public
D. private protected

33. The use of protected keyword to a member in a class will restrict its visibility as follows:
A. Visible only in the class and its subclass in the same package.
B. Visible only inside the same package.
C. Visible in all classes in the same package and subclasses in other packages
D. Visible only in the class where it is declared.

34. Which of the following are not keywords?
A. NULL
B. implements
C. protected
D. extended
E. string

35. Which of the following are keywords?
A. switch
B. integer
C. default
D. boolean
E. object

36. Which of the following keywords are used to control access to a class member?
A. default
B. abstract
C. protected
D. interface
E. public

37. The keywords reserved but not used in the initial version of Java are:
A. union
B. const
C. inner
D. goto
E. boolean
F. synchronized

38. Consider the following code:

```
class ClassA
{
    public static void main (String args [ ] )
    {
        ClassB b = classB ( ) ;
    }
    ClassA (int x) { }
}
class ClassB extends ClassA
{
}
```

What will happen when we compile and run this code?
A. Compile and run successfully
B. Error. ClassA does not define a no-argument constructor
C. Error. ClassB does not define a no-argument constructor
D. Error. There is no code in the class ClassB
E. Error. There is no code in the constructor ClassA (int x)

39. A package is a collection of
    A. classes B. interfaces
    C. editing tools D. classes and interfaces
40. Which of the following statements are true?
    A. An abstract class may not have any final methods.
    B. A final class may not have any abstract methods.
    C. An inner class may be declared with any accessibility keyword.
    D. Transient variables must be static.
41. Which of the following defines a legal abstract class?
    A. class Vehicle {
    abstract void display( ); }
    B. abstract Vehicle {
    abstract void display( ); }
    C. abstract class Vehicle {
    abstract void display( ); }
    D. class abstract Vehicle {
    abstract void display( ); }
    E. abstract class Vehicle {
    abstract void display( ); {
    System.out.println("Car"); }}
42. Package p1 contains the following code:

```
package p1;
public class Student { Body of student }
class Test { Body of Test }
```

Now consider the following code:

```
import p1.*;
class Result
{
    Student s1;
    Test t1;
}
```

This code will not compile because
A. Class Result should be declared public.
B. Student class is not available.
C. Test class is not available.
D. Result body is not fully defined.

43. Consider the following code:

```
interface Area
{
    float compute (float x, float y) ;
}
class Room implements Area
{
    float compute (float x, float y)
    {
```

```
        return (x & y) ;
    }
}
```

What is wrong with the code?

A. Interface definition is incomplete
B. Method **compute( )** in interface **Area** should be declared public
C. Method **compute( )** in class **Room** should be declared public
D. All the above

44. The concept of multiple inheritance is implemented in Java by
A. extending two or more classes
B. extending one class and implementing one or more interfaces
C. implementing two or more interfaces
D. all the above

45. Which of the following statements are valid array declaration?
A. int number( );
B. float average[ ];
C. double[ ] marks;
D. counter int[ ];

46. Consider the following code

```
int number [ ] = new int [5] ;
```

After execution of this statement, which of the following are true?
A. number[0] is undefined
B. number[5] is undefined
C. number[4] is null
D. number[2] is 0
E. number.length( ) is 5

47. What will be the content of array variable table after executing the following code?

```
for (int i=0; i<3; i++)
    for (int j=0, j<3; j++)
        if (j == i) table [i] [j] = 1;
        else table [i] [j] = 0;
```

A.
0 0 0
0 0 0
0 0 0

B.
1 0 0
1 1 0
1 1 1

C.
0 0 1
0 1 0
1 0 0

D.
1 0 0
0 1 0
0 0 1

48. Which of the following classes are available in the **java.lang** package?
A. Stack
B. Object
C. Math
D. Random
E. String
F. StringBuffer
G. Vector

49. Which of the following are the wrapper classes?
A. Random
B. Byte
C. Vector
D. Integer
E. Short
F. Double
G. String

50. Which of the following contain error?
A. int x[ ] = int[10];
B. int[ ] y = new int[5];

C. float d[ ] = {1, 2, 3};
D. x = y = new int [10];
E. int a[ ] = {1, 2}; int b[ ]; b = a;
F. int i = new int(10);

51. Which of the following methods belong to the **String** class?
A. length( ) B. compareTo( ) C. equals( )
D. substring( ) E. All of them F. None of them

52. Given the code

```
String s1 = "yes";
String s2 = "yes";
String s3 = new String (s1) ;
```

Which of the following would equate to **true**?
A. s1 = = s2 B. s1 = s2 C. s3 = = s1
D. s1.equals(s2) E. s3.equals(s1)

53. Suppose that s1 and s2 are two strings. Which of the statements or expressions are correct?
A. String s3 = s1 + s2; B. String s3 = s1 – s2; C. s1 <= s2
D. s1.compareTo(s2); E. int m = s1.length( );

54. Given the code

```
String s = new String ("abc") ;
```

Which of the following calls are valid?
A. s.trim( ) B. s.replace('a', 'A') C. s.substring(3)
D. s.toUpperCase( ) E. s.setCharAt(1,'A') F. s.append("xyz")

55. The methods **wait( )** and **notify( )** are defined in
A. java.lang.String B. java.lang.Runnable C. java.lang.Object
D. java.lang.Thread E. java.lang.ThreadGroup

56. Which of the following statements are true?
A. A Java monitor must either extend Thread or implement Runnable.
B. The sleep( ) method should be enclosed in try ... catch block.
C. The yield( ) method should be enclosed in try ... catch block.
D. A thread can be temporarily suspended from running by using the wait( ) method.
E. A suspended thread using suspend( ) method can be revived using the resume( ) method.

57. Given the following code:

```
class Base { int x = 10; }
class Derived extends Base
{ int x = 20; }
Base b = new Base ( ) ;
Derived d = new Derived( ) ;
Base bd = new Derived( ) ;
```

The statement

```
System.out.println (b.x + " " + d.x + " " + bd.x) ;
```

will produce the output
A. 10 20 20 B. 10 20 10
C. 20 10 20 D. 20 20 10

58. Given the class definitions

```
class Base
```

```
{
    void display( )
    { System.out.println ("Base") ; }
}
class Derived extends Base
{
    void display ( )
    { System.out.println ("Derived") ; }
}
```

and objects

```
Base b = new Base( );
Derived d = new Derived( );
Base bd = new Derived( );
```

then the print statements

```
System.out.print(b.display( ) + " ");
System.out.print(d.display( ) + " ");
System.out.print(bd.display( ) + " ");
System.out.println( );
```

will display:

A. Base Base Derived
B. Base Derived Base
C. Base Derived Derived
D. Derived Derived Derived

59. When we invoke **repaint( )** for a Component, the AWT invokes the method:
A. draw( ) B. show( )
C. update( ) D. paint( )

60. What does the following line of code do?

```
TextField text = new TextFiled(10);
```

A. Creates text object that can hold 10 rows of text.
B. Creates text object that can hold 10 columns of text.
C. Creates the object text and initializes it with the value 10.
D. The code is illegal.

61. Which of the following applet tags is legal to embed an applet class named Test into a Web page?
A. < applet
class = Test width = 200 height = 100>
< /applet>
B. < applet>
code = Test.class width = 200 height = 100>
< /applet>
C. < applet
code = Test.class width = 200 height = 100
< /applet>
D. < applet
param = Test.class width = 200 height = 100>
< /applet>

E. < applet
code = Test.class width = 200 height = 100>
< / applet>

62. Which of the following methods can be used to draw the outline of a square?
A. fillRect( ) B. drawLine( ) C. drawRect( )
D. drawString( ) E. drawPolygon( )

63. Which of the following methods can be used to change the size of a component
A. dimension( ) B. setSize( ) C. area( )
D. size( ) E. resize( )

64. Which of the following methods can be used to remove a component from the display?
A. delete( ) B. remove( ) C. disappear( )
D. hide( ) E. move( )

65. The setBackground( ) method is part of the class
A. Graphics B. Applet C. Component
D. Container E. Object

66. When we implement the Runnable interface, we must define the method
A. start( ) B. init( ) C. run( )
D. runnable( ) E. resume( ) F. main( )

67. Which of the following strings can be used as mode strings for creating a RandomAccessFile object?
A. " r " B. " w " C. " rw "
D. " wr " E. " 0 "

68. What will be the output of the following program?

```
class Main1
{
    public static void main(String args [ ])
    {
        boolean b = true;
        System.out.println("XXX");
        return;
        System.out.println("YYY");
    }
}
```

A. XXX
B. YYY
C. XXX followed by YYY
D. Error. Won't compile

69. What will be output of the following program?

```
class Main2
{
    public static void main(String args[ ])
    {
        boolean b = true;
        System.out.println("XXX");
        if( !b ) return;
        System.out.println("YYY");
    }
}
```

A. XXX
B. YYY
C. XXX followed by YYY
D. Error. Won't compile

70. DataInput is
A. an abstract class defined in java.io.
B. a class we can use to read primitive data types.
C. an interface that defines methods to open files.
D. an interface that defines methods to read primitive data types.

71. Which of the following statements are *true*?
A. Unicode characters are all 16 bits.
B. UTF characters are all 24 bits.
C. Reader class has methods that can read integers and floats.
D. File class may be used to rename a file.
E. DataOutputStream objects are used to write primitive data to a file.

72. Which are the valid ways to create DataInputStream streams?
A. new DataInputStream( );
B. new DataInputStream("in.dat", "r");
C. new DataInputStream("in.dat")
D. new DataInputStream(new File("in.dat"));
E. new DataInputStream(new FileInputStream("in.dat");

73. Which exception is thrown by the read( ) method of InputStream class?
A. Exception
B. FileNotFoundException
C. ReadException
D. IOException
E. None of the above

74. In the code below, what data types the variable x can have?

```
byte b1 = 5;
byte b2 = 10;
x = b1 * b2;
```

A. byte
B. int
C. short
D. long
E. float
F. double

75. If you want to assign a value of 99 to the variable year, then which of the following lines can be used within an <applet> tag?
A. number = getParameter(99)
B. < number = 99 >
C. < param = radius value = 99 >
D. < param name = number value = 99 >
E. < param number = 99 >

76. What is java_g used for?
A. Using the jdb tool
B. Executing a class with optimization turned off
C. To provide information about deprecated methods
D. None of the above

77. With javadoc, which of the following denotes a javadoc comment?
A. //#
B. /*
C. /**
D. //**

78. Given file is a File object, which of the following are legal statements to create a new file

A. file.create( );
B. FileOutputStream fos = new FileOutputStream(file);
C. FileWriter out = new FileWriter(file);
D. FileInputStream fis = new FileInputStream(file);
E. RandomAccessFile raf = new RandomAccessFile(file);

79. Which javadoc tage is used to denote a comment for a method parameter?
A. @method B. @parameter C. @argument
D. @param E. @value

80. Which of the following command lines options generates documentation for all classes and methods?
A. –protected B. –public C. –private
D. –verbose E. –encoding

81. Given the declarations

```
int x, m = 2000;
short y;
byte b1 = −40, b2;
long n;
```

which of the following assignment statements will evaluate correctly?
A. x = m * b1; B. y = m * b1;
C. n = m * 3L; D. x = m * 3L;

82. Given the declarations

```
boolean b;
short x1 = 100, x2 = 200, x3 = 300;
```

Which of the following statements are evaluated to *true*?
A. b = x1 * 2 = x2;
B. b = x1 + x2 != 3 * x1;
C. b = (x3 – 2*x2<0) || ((x3 = 400) <2**x2);
D. b = (x3 – 21*x2>0) || (x3 = 400) 2*x2);

83. In which of the following code fragments, the variable x is evaluated to 8.
A. int x = 32;'
x = x > > 2;
B. int x = 33;
x = x > > 2;
C. int x = 35;
x = x > > 2;
D. int x = 16;
x = x > > 1;

84. Consider the following code snippet:

```
.....
.....
try {
int x = 0;
int y = 50/x;
System.out.println("Division by zero");
}
catch(ArithmeticException e) {
System.out.println("catch block");
}
.....
.....
```

What will be the output?
A. Error. Won't compile
B. Division by zero
C. Catch block
D. Division by zero
Catch block

85. Which of the following represent legal flow control statements?
A. break; B. break( ); C. continue outer;
D. continue(inner); E. return; F. exit( );

Part C: Short-answer Questions

1. What will be the output of the following code?

```
byte x = 64, y;
y = (byte) (x << 2);
System.out.println(y);
```

2. What will be the output of the following code:

```
byte b;
double d = 417.35;
b = (byte) d;
System.out.println(b);
```

3. Given the value of a variable, write a statement, without using if construct, which will produce the absolute value of the variable.
4. What is wrong with the following code?

```
switch(x)
{
    case 1:
    n1 = 10;
    n2 = 20;
    case 2:
    n3 = 30;
    break;
    n4 = 40;
}
```

5. What will be the output of the following program code?

```
int m = 100;
int n = 300;
while (++m < --n);
System.out.println(m);
```

6. What does the following fragment display

```
String s = "six:" + 3 + 3;
System.out.println(s);
```

7. What is the output of the following code?

```
String s;
System.out.println("s = " + s);
```

8. What is the output of the following code?

```
String s = new String( );
System.out.println("s = " + s);
```

9. What is the problem with the following snippet?

```
class Q9
{
    public static void main(String args[ ])
    {
        int i = 5, j = 10;
        if ( (i<j) || (i=10) )
            System.out.println("OK");
        System.out.println("NOT OK");
    }
}
```

10. What will be the output of the following code snippet?

```
int x = 10;
int y = 20;
if ((x<y) || (x=5)>10)
    System.out.println(x);
else
    System.out.println(y);
```

11. Show the output the following code:

```
int a, b;
a = 5;
b = 10;
if(a > 5)
    if(b > 5)
    {
        System.out.println("b is " + b);
    }
    else
        System.out.println("a is " + a);
```

12. State the output of the following code:

```
int a = 10;
int b = 5;
if(a > b)
{
    if(b > 5)
    System.out.println("b is " + b);
}
else
    System.out.println("a is" + a);
```

13. Give the output of the following code:

```
int m = 100;
while(true)
{
```

```
    if(m < 10)
        break;
    m = m - 10;
}
Systm.out.println("m is " + m);
```

14. Give the output of the following code:

```
int m = 100;
while(true)
{
    if(m < 10)
    continue;
    m = m - 10;
}
System.out.println("m is " + m);
```

15. Using a single line of code, complete the following class so that it returns x+y if the value of x is equal to y, otherwise returns 0:

```
public class XY
{
    public return int fun(int x, int y)
    {
        ..... (one line code her)
    }
}
```

16. Given a package named **EDU.Student**, how would you import a class named **Test** contained in this package? Write one line statement.
17. Consider the following class definition:

```
class Student

{

    abstract double result( )

}
```

This code will not compile since a keyword is missing in the first line. What is the keyword?

18. Consider the following class file?

```
import java.awt.*;
import java.io.*;
package studentBase;
class Test
{
    void display( )
    {
        System.out.println("RESULTS");
    }
}
```

Will it compile? YES or NO. Give reason, if No:

19. Consider the following code:

```
class Product
{
    public static void main(String args [ ])
    {
        int x = 10, y = 20;
        System.out.println(mul (x, y));
    }
    int mul(int a, int b)
    {
        return(a * b);
    }
}
```

Will it compile? YES or NO. Give reason, if No:

20. Given below are two files:

File Employee.java

```
package purchase;
public class Employee
{
    protected double age = 35.00;
}
```

File Company.java

```
import purchase.Employee;
public class Company
{
    public static void main(String arg[ ])
    {
        Employee e = new Employee( );
        System.out.println("Age = " + e.age);
    }
}
```

Will the file Company.java compile? YES or NO. Give reason, if No.

21. Consider the following code:

```
class A
{
    void method(int x)
    { System.out.println("x = " + x); }
}
class B extends A
{
    void method(int y)
    { System.out.println("y = " + y); }
    void method(String s)
    { System.out.println("s = " + s); }
    public static void main(String args[ ])
    {
        A a1 = new A( );
```

```
        A a2 = new B( );
        a1.method(10);
        a2.method(20);
    }
}
```

What will be the output, when executed?

22. There are three classes that implement and **DataInput** and **DataOutput** interfaces. Two of them are **DataInputStream** and **DataOutputStream**. Which is the third one?
23. What output will the following program produce?

```
class Bits
{
    public static void main(String args[ ])
    {
        short s1 = 3; // 0000 0011
        short s2 = 13; // 0000 1101
        s1 = (short) (s1 ^ s2);
        System.out.println("Result is " + s1);
    }
}
```

24. State the output of the following program:

```
class Condition
    {
        public static void main(String args[ ])
        {
            int x = 10;
            int y = 15;
            System.out.println((x>y)? 3.14 : 3));
        }
    }
```

25. Which of the classes in **java.io** package defines a method to delete a file?
26. Given a valid File object reference, we can create a new file using two classes defined in **java.io** package. One is **FileOutputStream** class. Which is the other one?
27. If raf is an instance of **RandomAccessFile**, how can we move the file pointer to the end of the file? Write the statement.
28. What will be the output of the following program when it is executed with the command line java Command Java is wonderful

```
class Command
{
    public staitic void main(String args[ ])
    {
        for(int i = 1; i < args.length; i++)
        {
            System.out.print(args[i]);
            if( i != args.length )
            System.out.print(" ");
        }
```

```
            System.out.println(" ");
        }
    }
```

29. What will be the output of the following code snippet when combined with suitable declarations and run?

```
StringBuffer city = new StringBuffer("Madras");
StringBuffer string = new StringBuffer( );
string.append(new String(city));
string.insert(0, "Central ");
String.out.println(string);
```

30. Consider the following program code:

```
class Thread1 extends Thread
{
    public void run( )
    {
        System.out.println("Begin");
        suspend( );
        resume( );
        System.out.println("End");
    }
}
class ThreadTest
{
    public static void main(String args[ ])
    {
        Thread1 T1 = new Thread1( );
        T1.start( );
    }
}
```

On execution, what will be the output?

31. Consider the following application:

```
class Max
{
    public static void main(String args[ ])
    {
        int max = 10;
        max(max, 20, 30);
        System.out.println(max);
    }
    static void max(int max, int x1, int x2)
    {
        if(x1 > x2)
        max = x1;
    else
        max = x2;
    }
}
```

What value is printed out, when executed?

32. State the output of the following program:

```
class Recur
{
    public static void main(String args[ ])
    {
        int Result = result(10);
        System.out.println("Result = " + Result);
    }
    static int result(int m)
    {
        if (m <= 2)
            return m;
        else
            return m + result(m-2);
    }
}
```

33. Consider the class definition:

```
class Default
{
    public static void main(String args[ ])
    {
        int m;
        System.out.println("m is " + m);
    }
}
```

Will this code compile? YES or NO. Give reason, if No.

34. What is the output of the following program?

```
class Static
{
    static int m = 0;
    static int n = 0;
    public static void main(String args[ ])
    {
        int m = 10;
        int x = 20;
        {
            int n = 30;
            System.out.println("m + n = " + m + n);
        }
        x = m + n;
        System.out.println("x = " + x);
    }
}
```

35. Consider the following class definitions:

```
class Square
{
    private square( ) { }
```

```
        int area(int side)
        {
            return(side * side);
        }
    }
    class Constructor
    {
        public static void main(String args[ ])
        {
            Square S1 = new Square( );
            int area = S1.area(10);
            System.out.println(area);
        }
    }
```

Will the code above compile and run successfully. YES or NO. Give reason, if No.

36. Write a statement to draw a rounded rectangle with the following features:
width = 200
height = 100
corner horizontal diameter = 20
corner vertical diameter = 40
Select a suitable upper-left corner of the rectangle.

37. Which line of the following HTML file contains an error?

```
1.          < applet
2.                                          WIDTH = 400 HEIGHT = 200
3.                                          CODE = HelloJava.Class >
4.          < param
5.                                          NAME = "string"
6.                                          VALUE = "Hello" >
7.          </applet>
```

38. Give the output of the following program:

```
class MainString
{
    public static void main(String args[ ])
    {
        StringBuffer s = new StringBuffer("String");
        if(s.length()>5) &&
                                (s.append("Buffer").equals("X")
                                ; // empty statement
        System.out.println(s);
    }
}
```

39. What is the range of the value that can be assigned to a variable of type long?

40. Consider the following program :

```
class Number
{
     int x;
     void store(Number num)
     {
          num.x++;
     }
}
class MainNumber
{
     public static void main(String args[ ])
     {
          Number n = new Number( );
          n.x = 10;
          n.store(n);
          System.out.println(n.x);
     }
}
```

What is the output?

41. Given the code:

```
class Continue
{
     public static void main(String args[ ])
     {
     int m = 0;
     loop1: for(int i=0; i<10; i++)
          loop2: for(int j=0;j<10;j++)
               loop3: for(int k=0;k<10;k++)
               {
                    System.out.println(++m);
                    if( (k%10) == 0)
                         continue loop2;
               }
     }
}
```

What is the last value printed?

42. Can an abstract method by declared final? YES or NO. If NO, give reason.
43. Can an abstract method be declared static? YES or NO. If NO, give reason.
44. Consider the following **try ... catch** block:

```
class TryCatch
{
     public static void main(String args[ ])
     {
          try
          {
               double x = 0.0;
```

```
                throw(new Exception("Thrown"));
                return;
            }
            catch(Exception e)
            {
                System.out.println("Exception caught");
                return;
            }
            finally
            {
                System.out.println("finally");
            }
        }
    }
```

What will be the output?

45. Write a statement that would construct a 20 point bold Helvetica font.

ANSWERS

Part A: True/False Questions

1. A	2. B	3. B	4. A	5. B
6. A	7. A	8. A	9. B	10. B
11. B	12. B	13. A	14. B	15. A
16. B	17. A	18. B	19. A	20. B
21. B	22. B	23. A	24. B	25. B
26. B	27. B	28. A	29. A	30. B
31. A	32. B	33. A	34. A	35. B
36. B	37. A	38. B	39. B	40. A
41. B	42. A	43. A	44. B	45. B
46. A	47. B	48. B	49. A	50. B
51. B	52. A	53. B	54. B	55. A
56. B	57. B	58. B	59. A	60. B

Part B: Multiple-choice Questions

1. C	2. D & E	3. B & D	4. D	5. C
6. C	7. A, B, & C	8. C	9. B	10. C
11. B	12. B & C	13. D	14. C & E	15. D & E
16. C & B	17. E	18. A & C	19. A	20. B
21. A	22. B	23. D	24. C	25. C
26. B	27. C & D	28. A, C, & E	29. C	30. D
31. D	32. D	33. C	34. A, D & E	35. A & C
36. B, C & E	37. B, C & D	38. B	39. D	40. B & C
41. C	42. C	43. C	44. B & C	45. B & C
46. B, D & E	47. D	48. B, C, E & F	49. B, D, E & F	50. A, D, & F
51. E	52. A, D & E	53. A, D, & E	54. A, B, C & D	55. C
56. B, D & E	57. B	58. C	59. C	60. B

61.	E	62.	B, C & E	63.	B & E	64.	D	65.	C
66.	C	67.	A & C	68.	D	69.	C	70.	D
71.	A, D & E	72.	E	73.	D	74.	B, D, E & F	75.	D
76.	B	77.	C	78.	B, C & E	79.	D	80.	C
81.	A & C	82.	A & C	83.	A, B, C & D	84.	C	85.	A, C & E

Part C: Short answer Questions

1. 0
2. 161
3. x = x < 0? –x :x;
4. n = 40; is unreachable
5. 200
6. six : 33
7. null
8. s =
9. (i = 10) is the problem
10. 10
11. a is 5
12. No output
13. m is 0
14. No output; Infinite loop
15. retrun (x = = y)? x+y : 0;
16. import EDU.Student.Test;
17. abstract
18. No; The package definition must come first
19. No; The static method trying to invoke a non-static method
20. No; The field **age** in the Employee class should be public
21. x = 10; y = 20
22. RandomAccessFile class
23. Result is 14
24. Result = 3.0
25. File class
26. RandomAccessFile class
27. raf.seek(raf.length(( );
28. is wonderful
29. Central Madras
30. Begin
31. 10
32. 30
33. No
34. m + n = 40; x = 10
35. No
36. drawRoundRect (10,10,200,100,20,40);
37. Line 3
38. StringBuffer
39. -2^{63} to $2^{63} - 1$
40. 11
41. 100
42. No
43. No
44. Exception caught finally
45. new Font("Monospaced", Font.BOLD,20);

APPENDIX L

Points to Remember

L.1 GENERAL

1. It is important that the name of the file match the name of the class and the extension be **.Java.**
2. All functions in Java must be members of some class.
3. Member functions are called methods in Java.
4. Creating two methods with the same name but different arguments is called method overloading.
5. Method overloading allows set of methods with very similar purpose to be given the same name.
6. When a method makes an unqualified reference to another member of the same class, there is an implicit reference to **this** object.
7. Java does not provide a default constructor if the class defines a constructor of its own.
8. When present, **package** must be the first noncomment statement in the file.
9. The **import** statement must follow the **package** statement but must also precede all other noncomment statements.
10. The full method or class name, including the package name, must be used when two imported packages contain a method or class with the same name.
11. Due to security reasons, it is not possible to perform file I/O operations from an applet.
12. When a simple type is passed to a method, it is done by use of call-by-value. Objects are passed by use of call-by-reference.
13. It is illegal to refer to any instance variables inside of a **static** method.
14. All command-line arguments are passed as strings. We must therefore convert numeric values to their original forms manually.
15. A class member declared as private will remain private to its class. It is not accessible by any code outside its class, including subclasses.
16. The star form of import statement may increase compile time. It will be good practice to explicitly name the classes that we want to use rather than importing whole packages.
17. Interfaces add most of the functionality that is required for many applications which would normally require the use of multiple inheritance in C++.
18. When we implement an interface method, it must be declared as **public**.
19. If a **finally** block is associated with a try, the **finally** will be executed upon conclusion of the **try**.

20. Java uses pointers (addresses) internally to store references to objects, and for elements of any array of objects. However, these pointers are not available for use by programmers.
21. We cannot overload methods with differences only in their return type.
22. When a method with the same signature occurs in both the super class and its subclass, the method in the subclass overrides the method in the super class.
23. Every constructor must invoke its super class constructor in its first statement. Otherwise, the default constructor of the super class will be called.
24. A class marked as **final** cannot be inherited.
25. A method marked **final** cannot be overridden.
26. Subclasses of an abstract class that do not provide an implementation of an abstract method, are also abstract.

L.2 C/C++ RELATED

27. A Java string is not implemented as a null-terminated array of characters as it is in C and C++.
28. Most of the Java operators work much the same way as their C/C++ equivalents except for the addition of two new operators, >>> and ^.
29. The comparison operators in Java return a Boolean **true** or **false** but not the integer one or zero.
30. The modulo division may be applied to floating point values in Java. This is not permitted in C/C++.
31. The control variable declared in **for** loop is visible only within the scope of the loop. But in C/C++, it is visible even after the loop is exited.
32. Methods cannot be declared with an explicitly void argument list, as done in C++.
33. Java methods must be defined within the class. Separate definition is not supported.
34. Unlike C/C++, Java checks the range of every subscript and generates an error message when it is violated.
35. Java does not support the destructor function. Instead, it uses the **finalize** method to restore the memory.
36. Java does not support multiple inheritance.
37. C++ has no equivalent to the **finally** block of Java.
38. Java is more strictly typed than C/C++ languages. For example, in Java, we cannot assign a floating point value to an integer (without explicit type casting).
39. Unlike C/C++ which allow the size of an integer to vary based on the execution environment, Java data types have a strictly defined range and does not change with the environment.
40. Java does not support pointers.
41. Java supports labelled **break** and labelled continue statements.
42. **break** has been designed for use only when some sort of special situation occurs. It should not be used to provide the normal means by which a loop is terminated. ,
43. The use of **final** to a variable is similar to the use of **const** in C/C++.
44. Overridden methods in Java are similar to virtual functions in C++.
45. Java does not have a generalized console input method that parallels the **scanf** in C or **cin** in C++.
46. Integer types are always signed in Java. There are no unsigned qualifiers.
47. Java does not define escape codes for vertical tab and the bell character.
48. In Java multidimensional arrays are created as arrays of arrays. We can define variable size arrays of arrays.

APPENDIX M

Common Coding Errors

This appendix lists some coding errors that Java programmers are likely to make.

1. File name is not identical to the class name (in both spelling and case).
2. File not ending with **.Java** extension for a file containing the **main** method class (or applet's class definition).
3. Not ending a Java statement with a semicolon.
4. Providing spaces between the symbols of the operators = =, < =, > =, and !=.
5. Using operators < =, > =, and != as = <, = > and =!.
6. Using the operators = in the place of = =.
7. Using the keywords as identifiers.
8. Not properly matching the braces.
9. Placing a semicolon after the condition in an **if** statement or a **while** statement.
10. Using uppercase letters while writing a keyword.
11. Not initializing the variables properly.
12. Using floating point values in relational expressions.
13. Applying increment or decrement operator or an expression other than a simple variable.
14. Using commas instead of semicolons in a **for** header.
15. Placing a semicolon immediately after a **for** header.
16. Defining a method outside the braces of a class definition.
17. Not providing the return type in a method definition.
18. Not matching the type of the value returned by a method with its **return** type declared.
19. Placing a semicolon immediately after a method header.
20. Declaring a method parameter as a local variable inside the method.
21. Defining a method inside another method.
22. Attempting to assign a value to a **final** variable.
23. Using the same variable name in both the outer and inner blocks of a program.
24. Changing only return types for method overloading.
25. Referring to an element outside the array bounds.
26. Declaring a return type for a constructor method.
27. A class trying to access a **private** variable of another class.
28. Using **this** reference explicitly in a static method.

29. Trying to access an instance variable or an instance method inside a static method.
30. Assigning an object of a superclass to a subclass reference (without a cast).
31. Not using **super** method call as a first statement in the subclass constructor.
32. Attempting to instantiate an object of an **abstract** class.
33. Not declaring explicitly a class as **abstract** when it contains one or more **abstract** methods.
34. Using the instance variable **length** instead of method **length( )** to determine the length of a string.
35. Using the method **length( )** instead of the instance variable **length** to determine the size of an array.
36. Using string objects to access **StringBuffer** methods that are not members of the class **String**.
37. Attempting to access a character that is outside the bounds of a string.
38. Using a lower case f in **TextField**.
39. Trying to catch the same type of exception in two different **catch** blocks associated with a particular **try** block.
40. Placing **catch (Exception e)** before other **catch** blocks. This should be the last in the list of catch blocks.
41. Placing a **catch** that catches a superclass object *before* a **catch** that catches an object of its subclass.
42. Opening an existing file for output, when, in fact, the user wants to preserve the file.
43. Not opening a file before attempting to reference it in a program.
44. Using an incorrect stream object to refer to a file.

APPENDIX N

Glossary of Java Terms

abstract class	A class that cannot be instantiated directly. Abstract classes exist so that subclasses can inherit variables and methods from them.
access control	A way to restrict access to classes, variables and methods.
API	Application Programming Interface. The Java API contains classes a programmer can use to build applications and applets.
applet	A Java program that is embedded in an HTML document and runs in the context of a Java-capable browser.
appletviewer	A tool created by SUN to run applets without a browser.
argument	A value that is sent to a method when the method is called.
array	A list of values of the same type. All values in an array have a common name.
ASCII	A standard set of values for representing text characters.
assignment expression	Assigns a value to a variable.
attribute	A specifier for an HTML tag (for example, code is an attribute of the <APPLET> tag
AWT	The Abstract Windowing Toolkit, or group of classes for writing programs with graphical user interfaces.
baseclass	A class from which another class inherits functionality. In Java, a baseclass is often called a superclass.
bit	The smallest piece of data a computer understands. A bit can represent only two values, 0 or 1.
bitmap	A graphical image that is usually stored in a file.
Boolean	A value that can be either true or false.
Boolean expression	A Boolean expression evaluates to either true or false.
branching	When an execution jumps forward or backward in the program.

browser	A program used for reading, displaying, and interacting with objects on the World Wide Web.
byte	In Java, the byte is a data type, which is eight bits long.
bytecode	The machine-independent output of the Java compiler and input to the Java interpreter.
canvas	A applet component that can display graphics and text.
casting	Converting one type of value to another.
character	A value used in text. For example, the letters A-the digits 0-0 (when not used as mathematical values), spaces, and even tabs and carriage returns are all characters.
class	A collection of variables and methods that an object can have, or a template for building objects.
.class file	A file containing machine-independent Java bytecodes. The Java compiler generates .class files for the interpreter to read.
class variable	A variable allocated once per class. Class variables have global class scope and belong to the entire class instead of an instance.
client	A program that relies on services provided by another program called a server.
code	An attribute of the HTML <APPLET> tag that specifies the class to be loaded.
codebase	An attribute of the HTML <APPLET> tag that specifies the location of the classes to load.
comparison operators	Operators like = = (equals) and > (greater than) that compare two expressions, giving a result of true or false.
compiler	A language translator. A program that transforms source code into another format without executing the program.
concatenate	Adding one text string to the end of another.
conditional branching	When a program jumps to a different part of a program based on a certain condition being met.
configurable applet	An applet that the user can customise by supplying different parameters when writing the applet's tag in an HTML document.
constant	A value that never changes throughout the life of a program.
constructor	A method that is used to create an instantiation of a class.
control variable	The variable that a program evaluates to determine whether or not to perform an action. Control variables are used in loops, switch statements, and other similar programming constructs.
data field	The data that is encapsulated in an object.
data type	The type of value represented by a constant, variable, or some other program object. Java data types include the integer types byte, short, int, and long; the floating-point types float and double; the character type char; and the Boolean type boolean.
deadlock	Deadlock occurs when two or more threads are waiting for resources that they can't get.
derived class	A class that inherits from a base class.

dialog box	A special pop-up window that can present important information to the user or that requests information from the user. A dialog box is an object of Java's Dialog class.
doctags	Special symbols used by the javadoc tool to document Java packages and methods.
double	In Java, the double is a data type, which is 62 bits in length.
dynamic linking	When functions called within a program are associated with the program at runtime rather than at compile time.
encapsulation	A way to contain data and methods in a class so that methods and variables may be added, changed, or deleted without requiring the code that uses the class to change.
exception	A signal that something has happened to stop normal execution of a program, usually an error.
exception handler	Code that responds to and attempts to recover from an exception.
expression	A line of program code that can be reduced to a value or that assigns a value.
extends	A keyword used to make one class a subclass of another, for example, class subclass extends superclass.
field	A data object encapsulated in a class.
final	A modifier that prevents subclass definition, makes variables constant, and prevents a subclass from overriding a method.
finalize	A method that is called when there are no further references to an object and it is no longer needed. This method releases resources and does any other necessary cleanup that Java does not handle during garbage collection.
float	In Java, the float is a data type, which is thirty-two bits long.
floating point	A value with both whole number (including zero) and fractional parts.
font	A set of characters of the same style.
frame window	A special pop-up window that can be displayed from an applet. A frame window is an object of Java's Frame class.
GIF	One type of data format for storing graphical images on disk.
GUI	Stands for Graphical User Interface. It is pronounced like "gooey".
high-level language	A computer language that isolates the programmer from the intricate details of programming a computer. Java is a high-level language.
HotJava	A Java-capable browser from JavaSoft.
hspace	An attribute of the HTML <APPLET> tag that specifies the amount of horizontal space(to the left and right) between the applet and the text on the page.
HTML	Hypertext Markup Language, the language used to create Web pages.
identifier	A symbol that represents a program object.
index	The same as a subscript. Used to identify a specific array element.
infinite loop	A loop that cannot stop executing because its conditional expression can never be true.

inheritance	A property of object-oriented languages where a class inherits the methods and variables of more general classes.
initialise	Set the starting state of a program object. For example, you should initialise variables to a value before you attempt to use them in comparisons and other operations.
instance	A concrete representation of a class or object. A class can have many instances.
instance variable	A variable allocated once per instance of a class.
instantiate	To create a concrete object from a class "template". New objects are instantiated with new.
int	In Java, the int is a data type, which is 32 bits long.
integer	A whole-number value.
interface	A collection of methods and variables that other classes may implement. A class that implements an interface provides implementations for all the methods in the interface.
Internet	A huge world-spanning network of computers that can be used by anyone with a suitable connection.
Interpreter	A program that performs both language translation and program execution.java is the Java interpreter.
java	The program used to invoke the Java interpreter, which executes Java programs.
.java file	A file containing Java source code.
javac	A command for running the Java compiler.
javac_g	A command for running a non-optimized version of the Java compiler. The javac_g command can be used with debuggers, such as jdb.
Java-capable browser	A Web browser that can run Java applets. Also called a Java-enabled or Java-enhanced browser.
javadoc	A command that is used to generate API-style HTML documentation utomatically.
javah	A command that can create C include files and stubs from a Java .class file. The resulting C files allow C code to access parameters passed from Java, return values to Java, and access Java class variables.
javah_g	A command that can create C include files and stubs with debug information from a Java .class file.
javap	A command that disassembles Java .class files.
JavaScript	A Java-based scripting language.
jdb	The Java debugger.
JDBC	A database access API from JavaSoft that allows developers to access databases with Java programs.
JDK	The Java Developers Kit.
literals	Values, such as a number or text string, that are written literally as part of program code. The opposite of a literal is a variable.

local applet	An applet that is stored on the local computer system, rather than somewhere else on the Internet.
logical expression	An expression that results in a value of true or false, (see Boolean expression)
logical operators	Operators like && (AND) and || (OR) that enable you to create logical xpressions that yield true or false results.
long	In Java, the long is a data type which is 64 bits in length and can hold truly immense numbers.
loop	A program construct that enables a program to perform repetitive tasks.
method	A routine that belongs to a class.
modifier	A Java keyword that is applied to a method or variable declaration to control access, control execution, or provide additional information.
modular programming	Breaking a large program down into a number of functions, each of which performs a specific, well-defined task.
multidimensional array	An array that must be accessed using more than one subscript.
multiple inheritance	When a class simultaneously inherits methods and fields directly from more than one base class.
multitasking	Running more than one computer program concurrently (see multithread).
multithreaded	Having multiple threads of execution so that parts of a program can execute concurrently.
native methods	Methods that are declared in Java with the keyword native but are implemented in another language. Usually, native methods are written to do something that the Java API does not already do, to interact with a particular computer's hardware or operating system or to improve performance. Since native methods are not portable across platforms, applets cannot contain native methods.
nesting	When one program block is placed within another program block.
numerical expression	A combination of numbers, variables, or constants with operators.
Oak	The original name of the Java programming language.
object	An instantiation of a class.
object-oriented programming	A programming paradigm that treats program elements as objects that have data fields and functions that act on the data fields. The three main characteristics of OOP are encapsulation, inheritance, and the polymorphism.
one-dimensional array	An array that's set up like a list and requires only one subscript to identify a value contained in the array.
operator precedence	Determines the order in which mathematical operations are performed.
override	To replace a method inherited from a superclass.
package	A Java keyword used to assign the contents of a file to a package. Packages are Java's mechanism for grouping classes. Packages simplify reuse, and they are very useful for large projects.
parameter	A value that is passed to an applet or method. In the case of an applet, the parameters are defined in the HTML document, using the <PARAM> tag.

pass by reference	When an argument is passed into a method by passing a copy of the value. In this case, if you change the argument in the method, you also change the original.
pass by value	When an argument is passed into a method by passing a copy of the value. In this case, changing the copy doesn't affect the original value.
pixel	The smallest dot that can appear on the screen.
platform-neutral language	A programming language that can be used on any computer, regardless of the computer's operating system.
point	A unit of measurement of a font's height. One point is equal to 1/72 of an inch.
polymorphism	In object-oriented programming, this is the ability for a new object to implement the base functionality of a parent object in a new way.
protocol handler	A Java routine that interprets a protocol, generally for a browser.
radio buttons	A group of checkboxes in which only one checkbox can be selected at a time.
remote applet	An applet that is stored on another computer and which must be downloaded to the local computer over the Internet.
Runnable interface	An interface that allows a class the ability to run in a distinct thread without being a subclass of Thread.
runtime exception	An exception thrown by the system in response to unexpected problem when a program is running. An example would be the exception generated when an applet attempts to perform a division by zero.
scope	Defines where a method or variable is visible. A variable defined in a method is visible only.
server	A computer system that supplies services to another computer called client.
short	In Java, the short is a data type which is sixteen bits in length.
single inheritance	When a class inherits methods and fields directly from only one baseclass.
spaghetti code	Program code that keeps jumping from one place to another in the program without any apparent organisation.
standalone application	In the context of the Java language, an pan that doesn't need to be embedded in an HTML document. The opposite of an applet.
Streams	Controlled flows of data from one source to another. Java supplies several classes to create and manage streams. Classes that handle input data are derived from class InputStream, and classes that handle output data are derived from class OutputStream.
structured, programming	A style of programming in which the program code is divided into logically structured chunks of code.
stub	Part of the interface between Java code and a native method. A stub allows a native method to access Java parameters, access Java class variables, and return data to Java.
subclass	A class that inherits methods and variables from another class. The statement class ubClass extends Superclass means that SubClass is a subclass of Superclass.
subscript	A subscript is a number that identifies the element of an array in which a value is stored. A subscript is sometimes called an index.
superclass	A generalization of another class. X is a superclass of Y if Y inherits variables and methods from X.

symbolic constant	A word that represents a value in a program.
synchronized	A Java keyword that prevents more than one thread from executing inside a method at once.
tag	A command in an HTML document that identifies the type of document component.
thread	A single path of execution that is a subprocess of the main process. All applications have at least one thread, which represents the main program. An application can create additional threads to handle multiple tasks concurrently.
top-down programming	A style of programming that divides tasks up into general modules. At the top level of the program are only the general tasks, whereas, as we work out way deeper into the program code, the programming becomes more and more detailed.
two-dimensional array	An array that is set up much like a table, with a specific number of columns and rows.
type cast	Convert one type of value to another.
unconditional branching	When program execution jumps to a new part of the program regardless of any conditions.
Unicode	A new set of standard values for representing the symbols used in text. The Unicode character set is much larger than the ASCII character set and allows for foreign-language symbols.
unsigned	A value that can only be positive. This is the opposite of signed. Unsigned numbers are not used in Java programming.
URL	Stands for Uniform Resource Locator, which is an Internet address.
variable	A value that can change as often as necessary during the execution of a program. Variables are always represented by symbolic names.
virtual machine	An abstract, logical model of a computer used to execute Java bytecodes. A Java virtual machine has an instruction set, registers, a stack, a heap, and a method area.
VRML	Virtual Reality Modelling Language.
vspace	An attribute of the HTML <APPLET> tag that specifies the amount of vertical space above and below the applet and the text on the page.
Web browser	An application used to access the Internet's World Wide Web.
World Wide Web	The graphical part of the Internet.

APPENDIX O

Projects

I. NOTEPAD APPLICATION

O.01 LEARNING OBJECTIVES

The objectives of the Notepad application are as follows:

- To create a new text file
- To perform common operations on a text file, such as cut, copy and paste
- To demonstrate how to build a GUI application using the AWT package.

O.02 INTRODUCTION

The Notepad application works pretty much similar to the Notepad utility of Windows-based systems. It allows the users to create a new text document, add content to it, edit, save and close it. The Notepad application program given below uses the following classes:

- **Notepad:** Realizes the actual notepad.
- **gaListener:** Exits the applications.
- **Ne_option:** Creates a new text document.
- **Ope_option:** Opens the existing text document.
- **Clos_option:** Closes the Notepad application.
- **Sav_option:** Saves the current text document.
- **Cu_option:** Performs the cut operation.
- **Cop_option:** Copies the selected text within the currently open text document.
- **Past_option:** Pastes the text from the clipboard.

Notepad Application Program

```
import java.io.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.awt.*;
```

```
public class Notepad extends Frame
   {

   Clipboard cBoard = getToolkit().getSystemClipboard();
   TextArea tArea;
   String fName;

Notepad()
    {

    gaListener gListen = new gaListener();
    addWindowListener(gListen);

    tArea = new TextArea();
    add(tArea);

    MenuBar mBar = new MenuBar();
    Menu fileMenu = new Menu("File");

    MenuItem nOption = new MenuItem("New");
    MenuItem oOption = new MenuItem("Open");
    MenuItem sOption = new MenuItem("Save");
    MenuItem cOption = new MenuItem("Close");

    nOption.addActionListener(new Ne_option());
    fileMenu.add(nOption);

    oOption.addActionListener(new Ope_option());
    fileMenu.add(oOption);

    sOption.addActionListener(new Sav_option());
    fileMenu.add(sOption);

    cOption.addActionListener(new Clos_option());
    fileMenu.add(cOption);

    mBar.add(fileMenu);

    Menu editMenu = new Menu("Edit");
    MenuItem cutOption = new MenuItem("Cut");
    MenuItem copyOption = new MenuItem("Copy");
    MenuItem pasteOption = new MenuItem("Paste");

    cutOption.addActionListener(new Cu_option());
    editMenu.add(cutOption);

    copyOption.addActionListener(new Cop_option());
    editMenu.add(copyOption);

    pasteOption.addActionListener(new Past_option());
    editMenu.add(pasteOption);

    mBar.add(editMenu);
    setMenuBar(mBar);

setTitle("Notepad in Java");
}

  class gaListener extends WindowAdapter
```

```
  {
  public void windowClosing(WindowEvent closeNotepad)
    {
    System.exit(0);
  }
}
class Ne_option implements ActionListener
  {
  public void actionPerformed(ActionEvent ne)
    {
    tArea.setText(" ");

  }
}
class Ope_option implements ActionListener
  {
  public void actionPerformed(ActionEvent ope)
    {
    FileDialog fDialog = new FileDialog(Notepad.this,
    "Select a text file",FileDialog.LOAD);

    fDialog.show();

    if (fDialog.getFile()!=null)
    {
    fName = fDialog.getDirectory() + fDialog.getFile();
    setTitle(fName);
    ReadFile();
    }
    tArea.requestFocus();
  }
}
class Clos_option implements ActionListener
  {
  public void actionPerformed(ActionEvent close_o)
    {
    System.exit(0);
  }
}
class Sav_option implements ActionListener
  {
  public void actionPerformed(ActionEvent sav_o)
    {
    FileDialog fDialog = new FileDialog(Notepad.this,"Save the text file
    with .txt extension",FileDialog.SAVE);

    fDialog.show();

    if (fDialog.getFile()!=null)
      {
      fName = fDialog.getDirectory() + fDialog.getFile();
```

```
        setTitle(fName);
        try
        {
    DataOutputStream dOutStream = new DataOutputStream(new FileOutputStream(f
    Name));
      String oLine = tArea.getText();
      BufferedReader bReader = new BufferedReader(new StringReader(oLine));
while((oLine = bReader.readLine())!=null)
            {
            dOutStream.writeBytes(oLine + "\r\n");
            dOutStream.close();
          }
        }
        catch(Exception ex)
          {
          System.out.print("Required file not found");
        }
        tArea.requestFocus();
      }
    }
  }
  void ReadFile()
    {
    BufferedReader br;
    StringBuffer sBuffer = new StringBuffer();
    try
      {
        br = new BufferedReader(new FileReader(fName));
        String oLine;
      while((oLine=br.readLine())!=null)
        sBuffer.append(oLine + "\n");
        tArea.setText(sBuffer.toString());
        br.close();
    }
    catch(FileNotFoundException fe) { System.out.print("Required file not
    found");}
    catch(IOException ioe){}
  }
  class Cu_option implements ActionListener
    {
    public void actionPerformed(ActionEvent cut_o)
      {
      String sText = tArea.getSelectedText();
      StringSelection sSelection = new StringSelection(sText);
      cBoard.setContents(sSelection,sSelection);
      tArea.replaceRange
      ("",tArea.getSelectionStart(),tArea.getSelectionEnd());
```

```
    }
}
class Cop_option implements ActionListener
    {
    public void actionPerformed(ActionEvent copy_o)
        {
        String sText = tArea.getSelectedText();
        StringSelection cString = new StringSelection(sText);
        cBoard.setContents(cString,cString);
    }
}
class Past_option implements ActionListener
    {
    public void actionPerformed(ActionEvent paste_o)
        {
        Transferable ctransfer = cBoard.getContents(Notepad.this);
        try
            {
            String sText = (String)ctransfer.getTransferData(DataFlavor.stringFlavor);
tArea.replaceRange(sText,tArea.getSelectionStart(),tArea.getSelectionEnd());
        }
        catch(Exception exc)
            {
            System.out.println("Not a string flavor");
        }
    }
}
    public static void main(String args[])
        {
        Frame nFrame = new Notepad();
        nFrame.setSize(600,600);
        nFrame.setVisible(true);
    }
}
```

O.03 RUNNING THE APPLICATION

To run the Notepad application, we need to perform the following steps:

1. Run the following command at the command prompt:

```
javac Notepad.java
```

2. After successful compilation, run the following command:

```
java Notepad
```

3. The Notepad application gets launched, as shown in Fig. 0.1:

Fig. O.1 *The output of Notepad application*

Now, have can perform text-based tasks just like any other text-editor.

II. SKETCHPAD APPLICATION

O.04 LEARNING OBJECTIVES

The objectives of the Sketchpad application are as follows:

- To draw various geometric shapes (square, rectangle, circle)
- To fill different colours in geometric shapes
- To demonstrate how to build a GUI application using the AWT package

O.05 INTRODUCTION

The sketchpad application allows the users to create common geometric shapes using mouse. The Sketchpad application program given below implements methods of the following interfaces to perform sketching operations:

- ActionListener
- WindowListener
- MouseListener
- MouseMotionListener
- ItemListener

Sketchpad Application Program

```
   import java.awt.event.*;
import java.awt.*;

class sketch_pad extends Frame implements ActionListener,WindowListener,Mouse
Listener,MouseMotionListener,
ItemListener
{
   String selected_shape = new String("Square");

   String selected_color = new String("Blue");

   boolean Eraser=false;

   int up_L_X, up_L_Y, W, H, sel_x1,sel_y1,sel_x2,sel_y2;

   String[] extras_list = {"Clear Canvas","Eraser"};

   String[] color_list = {"Black","Cyan","Green","Yellow","Magenta","Red","Blue"};

   String[] shape_list = {"Line","Rectangle","Square","Circle"};

   public void windowClosing(WindowEvent eve){ System.exit(0);

   }
   public void windowActivated(WindowEvent eve){}
   public void windowOpened(WindowEvent eve){}
   public void windowIconified(WindowEvent eve){}
   public void windowClosed(WindowEvent eve){}
   public void windowDeactivated(WindowEvent eve){}
   public void windowDeiconified(WindowEvent eve){}
   public void mouseMoved(MouseEvent mouse_mov_eve){}
   public void mouseClicked(MouseEvent mouse_clicked_eve){}
   public void mouseExited(MouseEvent mouse_exited_eve){}
   public void mouseEntered(MouseEvent mouse_entered_eve){}
   public void itemStateChanged(ItemEvent item_state_chng_eve){}

   public sketch_pad(String str)
   {
     super(str);
     addMouseMotionListener(this);
     addWindowListener(this);
       addMouseListener(this);
       setLayout(null);
     set_menu_items();
     setBackground(Color.white);
   }
```

```
    public void actionPerformed(ActionEvent action_performed_eve)
    {
      Graphics ga = getGraphics();
      Object s = action_performed_eve.getActionCommand();
      for (int i=0; i != color_list.length; i++)
        if (s.equals(color_list[i]))
        {
          selected_color = color_list[i];
          return;
        }
      for (int i=0; i != shape_list.length; i++)
          if (s.equals(shape_list[i]))
          {
            selected_shape = shape_list[i];
            return;
          }
      if (s.equals("Eraser"))
      {
          Eraser = true;
          return;
      }
      else if (s.equals("Clear Canvas"))
      {
          ga.clearRect(0,0,700,700);
          return;
      }
    }
    void choose_color(Graphics ga)
    {
      for (int i=0; i!= color_list.length; i++)
      {
        if (selected_color.equals(color_list[i]))
        {
          switch (i)
          {
case 0: ga.setColor(Color.black);break;
case 1: ga.setColor(Color.cyan);break;
case 2: ga.setColor(Color.green);break;
case 3: ga.setColor(Color.yellow);break;
case 4: ga.setColor(Color.magenta);break;
case 5: ga.setColor(Color.red);break;
case 6: ga.setColor(Color.blue);
          }
        }
      }
    }

    public void mouseReleased(MouseEvent mouse_reles_eve)
    {
      Graphics ga = getGraphics();
```

```
if(Eraser)
{
  Eraser = false;
  return;
}
choose_color(ga);
sel_x2=mouse_reles_eve.getX();
sel_y2=mouse_reles_eve.getY();
if (selected_shape.equals("Line"))
{
  ga.drawLine(sel_x1,sel_y1,sel_x2,sel_y2);
}
else if (selected_shape.equals("Circle"))
{
  draw_selected_shape(ga,"Circle");
}
else if (selected_shape.equals("Square"))
{
  draw_selected_shape(ga,"Square");
}
else if (selected_shape.equals("Rectangle"))
{
  draw_selected_shape(ga,"Rectangle");
}
ga.setColor(Color.yellow);
ga.drawString(".",sel_x1,sel_y1);
ga.setColor(Color.black);
}
void draw_selected_shape(Graphics ga,String sel_shape)
{
  up_L_X = Math.min(sel_x1,sel_x2);
  up_L_Y = Math.min(sel_y1,sel_y2);
  W = Math.abs(sel_x1-sel_x2);
  H = Math.abs(sel_y1-sel_y2);
  if (sel_shape.equals("Square") )
    ga.fillRect(up_L_X,up_L_Y,W,W);
    else if (sel_shape.equals("Rectangle"))
    ga.fillRect(up_L_X,up_L_Y,W,H);
  else if (sel_shape.equals("Circle") )
    ga.fillOval(up_L_X,up_L_Y,W,W);
}
public void mouseDragged(MouseEvent mouse_drag_eve)
{
  Graphics ga = getGraphics();
  sel_x2=mouse_drag_eve.getX();
  sel_y2=mouse_drag_eve.getY();
  if (Eraser)
  {
    ga.setColor(Color.white);
```

```
            ga.fillRect(sel_x2,sel_y2,10,10);
          }
        }
        public void mousePressed(MouseEvent mouse_press_eve)
        {
          if (Eraser) return;
          up_L_X=0; up_L_Y=0; W=0; H=0;
          sel_x1=mouse_press_eve.getX();
          sel_y1=mouse_press_eve.getY();
          Graphics ga = getGraphics();
          ga.drawString(".",sel_x1,sel_y1);
        }
      void set_menu_items()
        {
          MenuBar mBar = new MenuBar();

            Menu menu_sh = new Menu("Shapes");
          for (int i=0; i != shape_list.length; i++)
            menu_sh.add(shape_list[i]);
          mBar.add(menu_sh);
          menu_sh.addActionListener(this);

            Menu menu_col = new Menu("Colors");
          for (int i=0; i != color_list.length; i++)
            menu_col.add(color_list[i]);
          mBar.add(menu_col);
          menu_col.addActionListener(this);

          Menu Ex = new Menu("Extras");
          for (int i=0; i != extras_list.length; i++)
            Ex.add(extras_list[i]);
          mBar.add(Ex);
          Ex.addActionListener(this);
       setMenuBar(mBar);
        }
    }

    class Sk_pad
    {
      public static void main(String[] args)
      {
          sketch_pad draw_win = new sketch_pad("Sketchpad in Java");
          draw_win.setSize(700,700);
          draw_win.setVisible(true);
      }
    }
```

O.06 RUNNING THE APPLICATION

To run the Sketchpad application, we need to perform the following steps:

1. Run the following command at the command prompt:

```
javac Sk_pad.java
```

2. After successful compilation, run the following command:

```
java Sk_pad
```

3. The sketchpad application gets launched, as shown in Figure O.2:

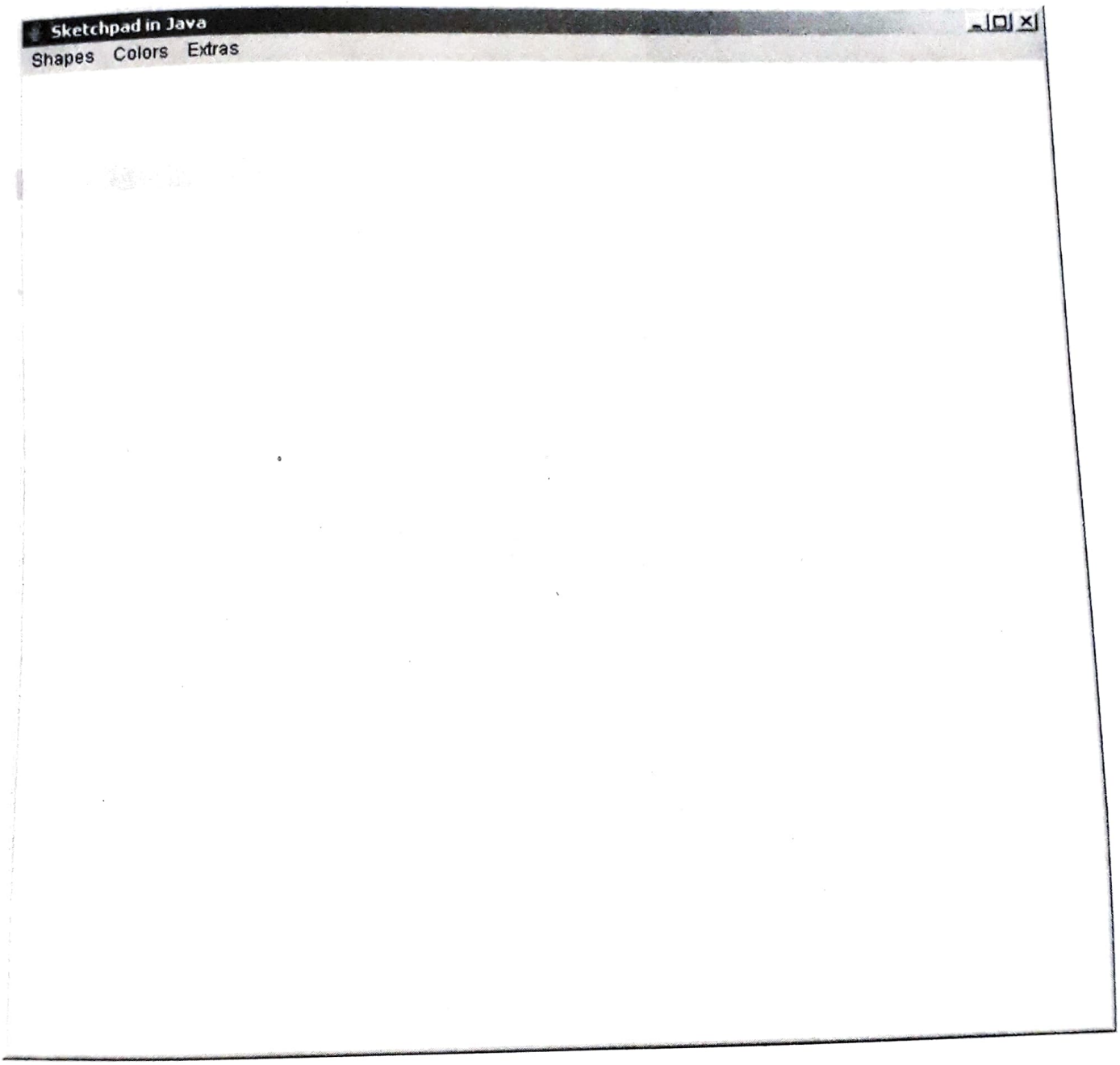

Fig. O.2 *The output of Sketchpad application*

Now, we can work on the Sketchpad application by using mouse to sketch different geometric shapes, as shown in Figure O.3:

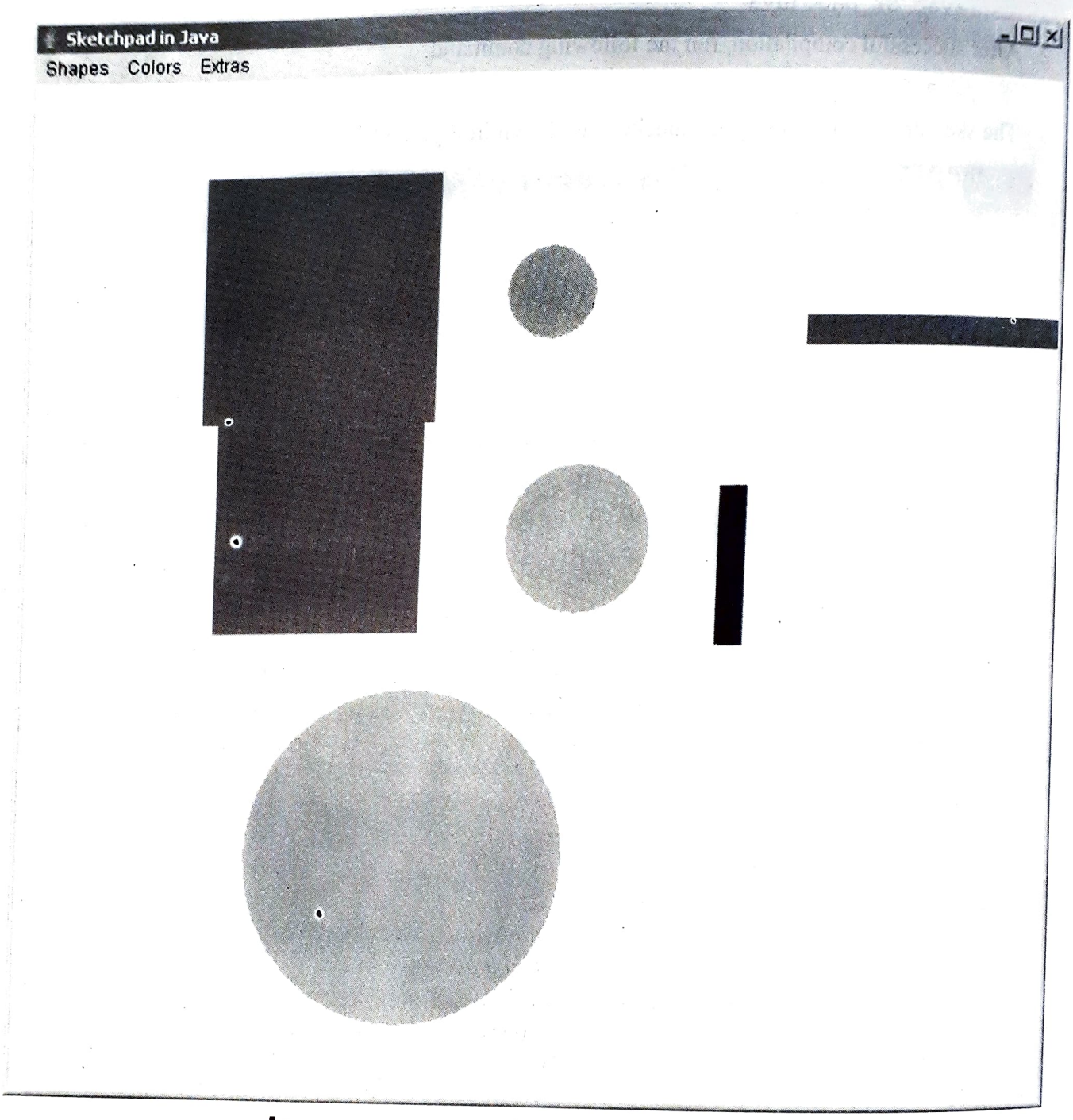

Fig. O.3 *Drawing shapes in Sketchpad*