SUB{}ASH PROGRAMMING CLASSES™
CODING BEYOND CURRICULUM

presents

Java
The Master Course

Subhash Programming Classes
Bangalore

www.subhashprogrammingclasses.in

+91 9845456000 / 9739156389

Compiled by

Subhash.K.U
Principal Mentor,
Subhash Programming Classes
Bangalore

About **Subhash Programming Classes**:

Subhash Programming Classes is India's only academy to train students and working professionals in the area of programming languages thoroughly, practically and professionally. Established in the year 2009 by our Principal Mentor, **Mr.Subhash.K.U**, we have currently trained more than 25,000+ students and placed 20,000+ students successfully across the globe. You can find our students in almost all the IT companies as Software Programmers. Our students say that, 95% of all interview questions comes from our classes. After attending our classes, we are sure, you cannot fail in any interview. All Embedded Systems related trainings are carried out under the banner **Subhash Embedded Classes.**
Visit: **www.subhashembeddedclasses.in**

About **Subhash.K.U**:

Subhash.K.U is currently the principal mentor at **Subhash Programming Classes**. He is a very passionate teacher and programmer who loves exploring programming languages. Having worked in the industry as Software Engineer, Senior Software Engineer, Consultant Engineer, Senior Design Engineer, he completely switched towards his passion to teach in the year 2011 and from then on there is no looking back. He is India's most liked trainer and till date trained more than 25,000+ students and personally placed thousands of students. You can reach him @ **subhashclasses@gmail.com** or **www.facebook.com/kusubhash.**

How to join Subhash Programming Classes?

1. Visit www.subhashprogrammingclasses.in
2. Or Visit www.subhashembeddedclasses.in
3. Select a course.
4. Call to +91 9845456000.
5. Enrol for your favourite course.

Why should I join Subhash Programming Classes?

1. We love to teach. We are passionate teachers.
2. Learn from India's top class trainers.
3. We make you an expert in the subject.
4. You can crack any programming interview.
5. Stay ahead than others with your magical and amazing technical skills.

Our Challenge:

"95% of all interview questions comes from our classes"

Our Greatest Achievement

6+ years of existence
25000+ students mentored
20000+ placements
300+ companies
1 teacher - Subhash.K.U

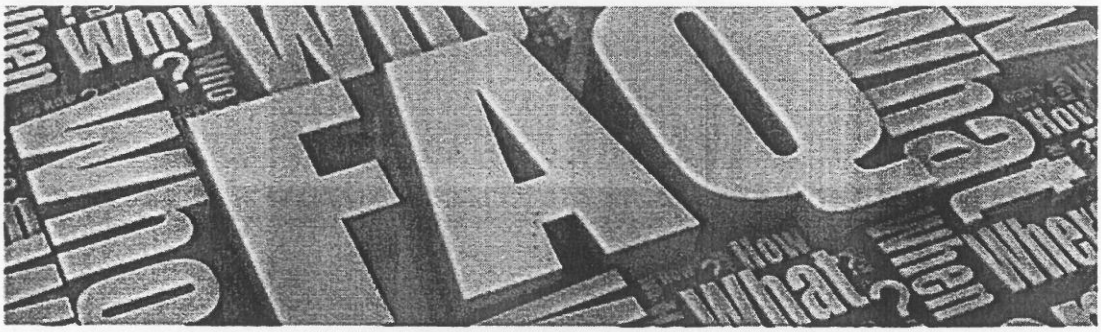
Our Goal

Place all our students as Professional Programmers in the IT industry

References used for making this study material:

1. The Java Language Specification - Bill Joy, et al.
2. Java - The Complete Reference - Herbert Schildt
3. Java In a Nutshell - Benjamin J Evans
4. Beginning Programming with Java For Dummies - Barry Burd
5. Java: The Best Guide to Master Java Programming Fast - Andrew Hoffman
6. Think Java - Allen B. Downey, et al.
7. Learning Java - Patrick Niemeyer, et al.
8. Head First Java - Kathy Sierra, et al.
9. Core Java - Cay.S.Horstman, et al.
10. Introduction to Java Programming - Liang
11. Effective Java - Joshua Bloch
12. Java Concurrency in Practice - Joshua Bloch
13. Thinking in Java - Bruce Eckel

All the cartoon images in this material has been taken from the world wide web. Thanks to all those creative designers who are the original creators of those cartoon images. I don't own any of the cartoon images.



- How to become programming geek?
- How to excel in IT industry
- How to find a good job?
- How do I know I am good in programming?
- How to increase my programming skills?
- Which is the best company to work?
- Which company pays more?
- Is salary an important criterion for a fresher?
- What do you mean by big companies and small companies
- Is programming a tough job?
- How many programming languages do I need to learn?
- How many books do I need to learn or read?
- What is the time frame required to learn programming?
- Will my low aggregate marks be an obstacle for my job search?
- Can I become a professional programmer with no communication skills?
- Which is the best course that fetches a job easily?

Do you have any other question??? We are happy to guide you!!

What we expect from students of SUBHASH PROGRAMMING CLASSES ?

- Commitment to the course. Commitment is not promise.
- An enthusiastic learning attitude.
- Regular attendance.
- Regular and timely assignment completion.
- Practical way of learning (no by-heart learning and no marks hungry learning).
- Create friendly environment. Make more friends in class.
- Teach others - Actual way of learning.
- Be happy to learn from unknowingly committed mistakes. No learning without mistakes.
- Work long hours. Train your mind. Meditate. Visualise.
- Update yourself on latest technologies.
- Feel free to talk to mentors for any concerns.
- Openness to ask doubts. Feel free.
- Read classical books suggested by mentors.

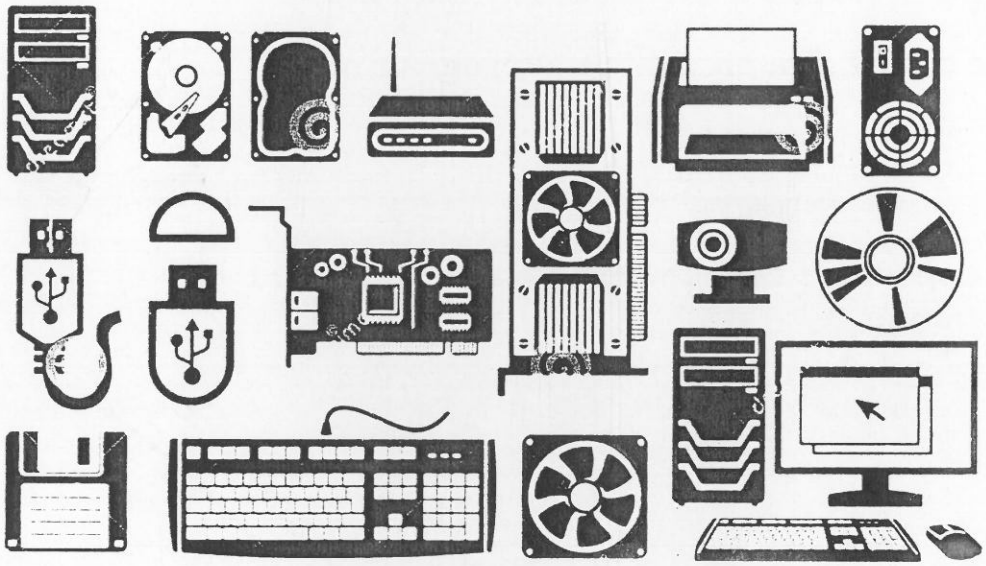
KEEP
CALM
AND
LEARN TO
CODE

A Quick Refresher

+

General Knowledge !

Let us discuss the basics first !



- A computer
  - CPU
    - Speed measured in terms of Hertz(Hz) - 1 Hz equals to 1 pulse/sec.
  - Memory (Main Memory) / RAM
    - CPU's work area that can be addressed to store programs & data
  - Storage Devices
    - Hard Disks, USB drives, CDs
  - Input Devices

- Keyboards, Mouse etc.

- Output Devices
  - Monitor, Printers

- Communication Devices
  - NIC, Modem etc.

Give few examples of Programming Languages.

- C, C++, Java, C# etc.

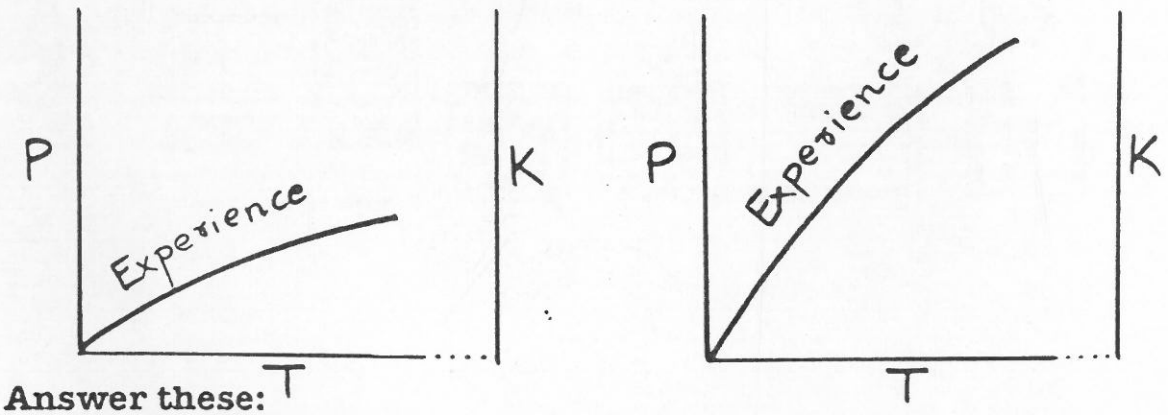
Give few examples of operating systems.

- Apple's Mac OS, GNU/Linux, Microsoft Windows

Who is called as a professional programmer ?

- Good Conceptual knowledge
- Good at Logical Thinking
- Good at Debugging

How long will it take to become a professional programmer ?



Answer these:

1 kg = _____ 1 ltr = _____ 1 mtr = _____ 1 byte = _____

What is the biggest data that you can store using:

1 byte = _____ 2 bytes = _____ 4 bytes = _____

How do you represent the following in binary?

2 = _____ -2 _____

What is System Software & Application Software?

- OS, Compilers, Interpreters etc (System Software)
- BookMyShow.com , MakeMyTrip.com , flipkart.com

Which is the best programming language ?

- Only immature fellows would ask these questions.

What are the different domains in an IT industry ?

- Embedded, Telecom, Healthcare, E-Commerce, Finance, Banking, Automative, etc.

How many programming languages to learn ?

- 1 or 2 PLs and 1 OS thoroughly.
- Do not become a "**Jack of all and master of none**" (Dangerous)

Exercise:

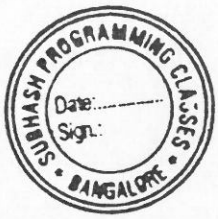
- Visit any PC Sales and Service Centre. Mandatory !!

The Beginning !!!

(Convince your mind - You wont be able to understand everything today - Just move on)

Credits & Appreciation:

- James Gosling & Team at Sun Microsystems

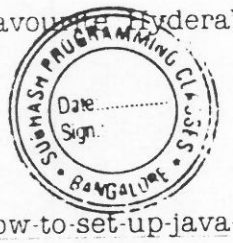


What is the difference between C and Java ?

- System Programming - C
- Application Programming - Java

What is the first step to learn Java ?

OK ! What is the first step to cook your favourite Hyderabadi Vegetable Biriyani ?



How to install and set up JDK ?

http://www.subhashprogrammingclasses.in/how-to-set-up-java-run-time-environment-for-beginners/

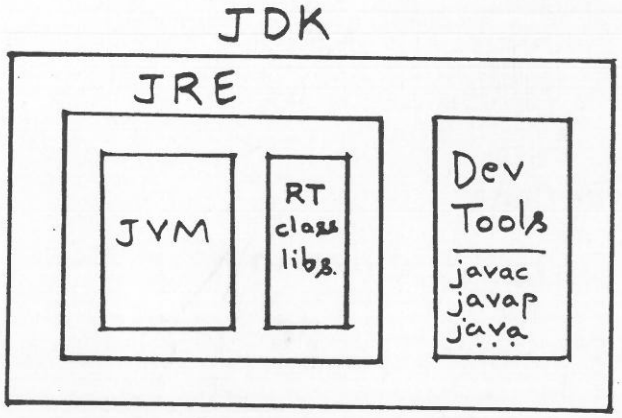
Digest it slowly: Java Language Specification, Java API, JDK and IDE:

- Java Language Specification - Rules of the language (Syntax) (https://docs.oracle.com/javase/specs/)
- API - Application Programming Interfaces (Pre-defined classes and interfaces for developing Java Programs. (www.oracle.com/technetwork/java/index.html)
- JDK - Java Development Kit (Set of programs/softwares each invoked from a command line, for developing and testing Java Programs)
- IDE - A Software that provides an integrated development environment for developing Java Programs quickly. Editing, compiling, building, debugging and online help are integrated in one graphical user interface.

Digest it slowly:

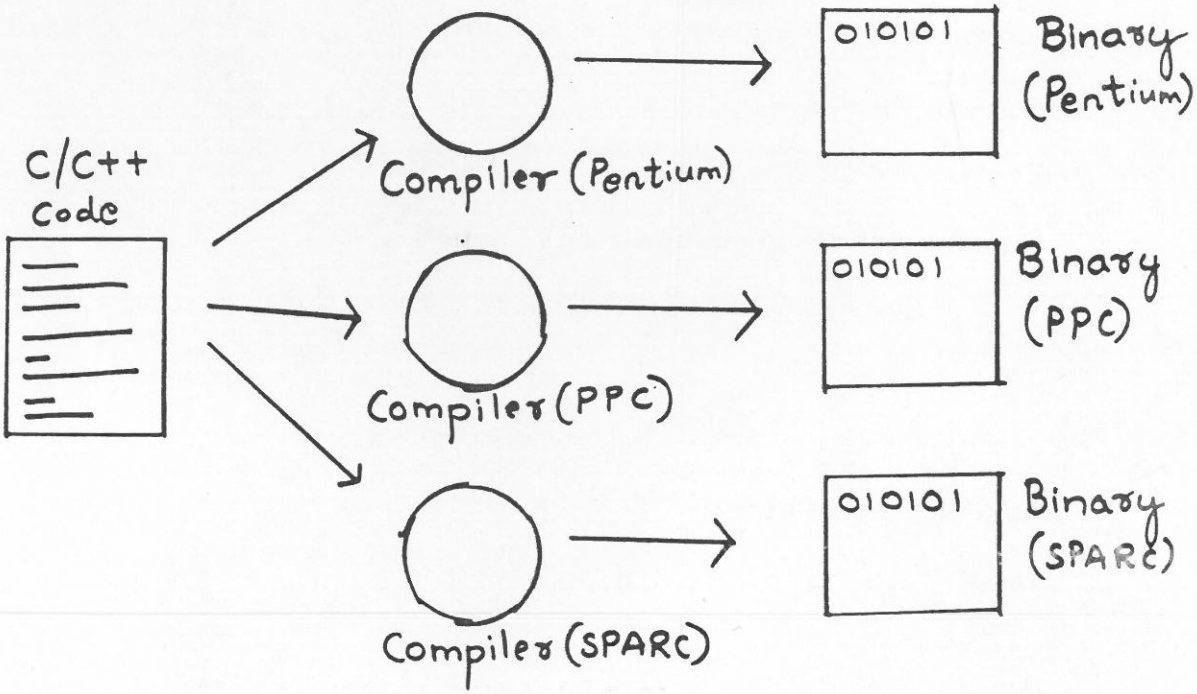
Java SE : To develop stand-alone client apps/applets.
Java EE : To develop server side apps like Servlets, JSP/JSF.
Java ME: To develop mobile apps for cell phones

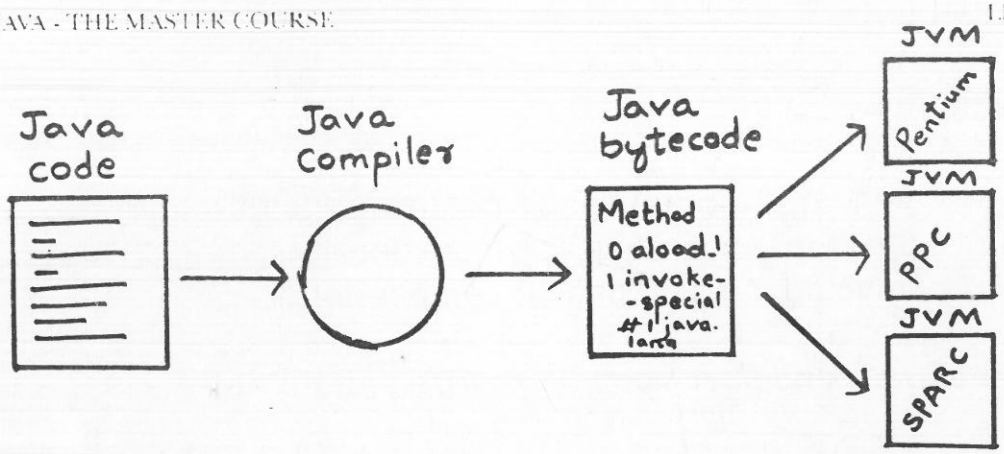
What is the difference between JRE, JDK, and JVM ?



JDK = JRE + Dev. T.
↓
JVM + libs.

What is the difference between C/C++ Compilation and Java Compilation ?





Steps to compile a Java Source Code:

→ Type your source code – className.java
→ Compile your source code – use "javac"
→ Run the "className.class" – use "java"

My First Java Program:

```
/* This is my First Java Program */

public class MyFirstJavaProgram
{
    public static void main( String [] args )
    {
        System.out.println( "Hello Java, I wish to marry you !" );
    }
}
```

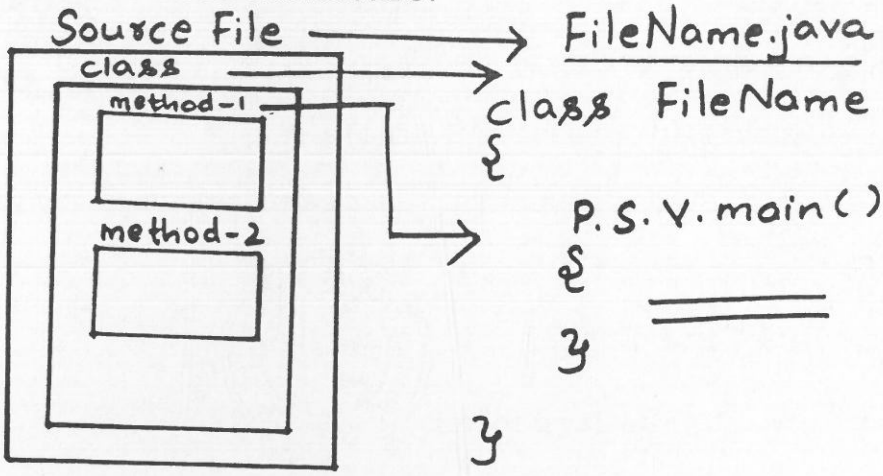
- Save the source file with the same name as your class name.

How to Compile and Run ?

```
$ javac MyFirstJavaProgram.java
$ java MyFirstJavaProgram

Hello Java, I wish to marry you !
```

Let us re-visit the Java compilation process discussed above.

General Java Source Code Structure:



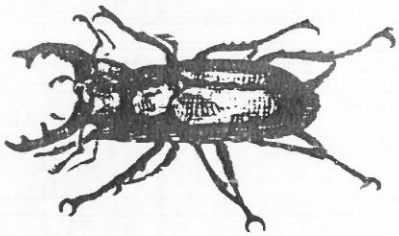
Few more sample source codes:

```
public class PrintThreeMessages
{
        public static void main( String [] args )
        {
                System.out.println( "Subhash " );
                System.out.println( "Programming " );
                System.out.println( "Classes " );
        }
}
```

```
public class PrintThreeMessagesOnSameLine
{
        public static void main( String [] args )
        {
                System.out.print( "Subhash " );
                System.out.print( "Programming " );
                System.out.print( "Classes " );

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

```
public class DoMathematicalOperations
{
        public static void main( String [] args )
        {
                System.out.println( (10.5 + 2 * 3) / ( 45 - 3.5 ));
        }
}
```

Programming Errors / Bugs:



1. Compile-Time Errors

```
public class MyFirstJavaProgram
{
        public static void main( String [] args )

                System.out.println( "Hello Java, I wish to marry you !" );
        }
}
```

2. Run-Time Errors

```
public class MyFirstJavaProgram
{
        public static void main( String [] args )
        {
                System.out.println( 8 / 0 );
        }
}
```

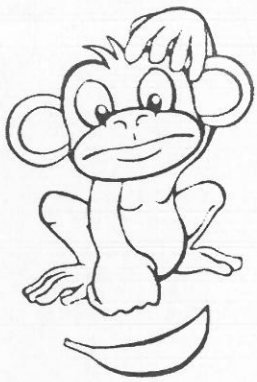
3. Logical Errors

```
public class MyFirstJavaProgram
{
        public static void main( String [] args )
        {
                System.out.println( (5/9) * ( 45 - 32 ) );
        }
}
```

Programming Style and Documentation:

Anyone can judge you based on your Coding and Documentation Style.

Guess the Output:



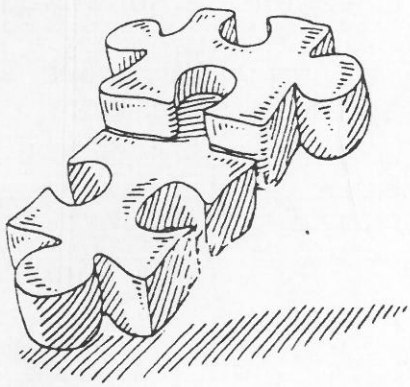
1.

```
public class GuessTheOutput

{
        public static void main( String [] args )
        {
                System.out.println( "5 * 4 / 4 - 3.5" );
                System.out.println( 5 * 4 / 4 - 3.5 );
                System.out.println( "Subhash" );
                System.out.println( 5 + "Subhash" );
                System.out.println( "Subhash" + 5 );
                System.out.println( 5 + 5 + "Subhash" + 5 );
                System.out.println( 5 + "Subhash" + 5 + "Subhash" );
                System.out.println( "Subhash" + 5 + 5 );
        }
}
```

Try this out:

1. Re-arrange the following code to print "one" followed by "two":

```
}
public class ReArrangeCode {
        System.out.println( "two" );
public void main( String [] args ) {
                System.out.println( "one" );
}
```



- Install JDK for your system. (**Refer**: www.subhashprogramming-classes.in/blog)

- Install NetBeans / Eclipse on your system. (If you wish to work on it)
- Practice all programs given in the material till now.
-
- WAP that displays WELCOME TO SUBHASH PROGRAMMING CLASSES and YOU HAVE SUCCEEDED 5 times.

- Try to purposefully make mistakes and check for various compile time and run-time errors.

All About Types !

```
public class AddIntAndInt
{
    public static void main( String [] args )
    {
        int numOne, numTwo, sum;

        numOne = 5;
        numTwo = 6;

        sum = numOne + numTwo;

       System.out.println( "Sum of " + numOne + " and " + numTwo + " = " + sum );
    }
}
```

Variables can be either of two types:

a. Primitive:
   boolean char byte short int long float double

b. Reference:
   Discussed later.

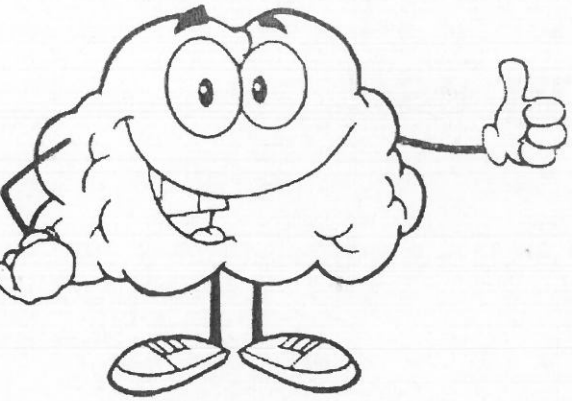
I would suggest you to use "**camelCase**" naming convention

The rules for naming a variable are as follows:

- You cannot start a name with a number. It must start with a letter, number or a dollar ($) sign.

- You can name a variable anything you wish to as long as it is not a Java reserved Keyword.

- WAP to add two floating point numbers.
- WAP to add one int and one float numbers.



WARNING:

Do not see others code while writing your own code. Do it yourself. Flex your brain. Brain will be delighted !!

Generalizing the code by I/O:

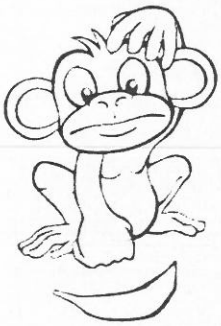

```
import java.util.*;
public class AddIntAndInt
{
        public static void main( String [] args )
        {
                int numOne, numTwo, sum;
                Scanner in = new Scanner( System.in );
                numOne = in.nextInt( );
                numTwo = in.nextInt( );
                sum = numOne + numTwo;
                System.out.println( "Sum of " + numOne + " and " + numTwo + " = " + sum );
    }
}
```

All in one:

```
public class ScanThroughKeyBoard
{
    public static void main( String [] args )
    {
        int iNumOne, iNumTwo, iSum;
        double dNumOne, dNumTwo, dSum;
        float fNumOne, fNumTwo, fSum;

        System.out.println( "Enter 2 integer numbers" );

        Scanner in = new Scanner( System.in );
        iNumOne = in.nextInt( );
        iNumTwo = in.nextInt( );
        in.nextLine( );
        iSum = iNumOne + iNumTwo;
        System.out.println( "Sum of " + iNumOne + " and " + iNumTwo + " = " + iSum );


        System.out.println( "Enter 2 double numbers" );
```

```
        dNumOne = in.nextDouble( );
        dNumTwo = in.nextDouble( );
        in.nextLine( );
        dSum = dNumOne + dNumTwo;
        System.out.println( "Sum of " + dNumOne + " and " + dNumTwo + " = " + dSum );

        System.out.println( "Enter 2 float numbers" );
        fNumOne = in.nextFloat( );
        fNumTwo = in.nextFloat( );
        in.nextLine( );
        fSum = fNumOne + fNumTwo;
        System.out.println( "Sum of " + fNumOne + " and " + fNumTwo + " = " + fSum );

        System.out.println( "Enter a boolean" );
        boolean yn = in.nextBoolean( );
        System.out.println(yn);
        in.nextLine( );

        System.out.println( "I will teach you how to read a string from an input" );
        String line = in.nextLine( );
        System.out.println( line );

       System.out.println( "I will teach you how to fetch only a word from an input" );
        String word = in.next( );
        System.out.println( word );

        in.nextLine( );
       System.out.println( "I will teach you how to fetch only a single character from
an input" );
        char character = in.next().charAt(0);
        System.out.println( character );

    }
}
```


- WAP to find the area of a triangle inputting three sides.
- WAP to find the area of a circle inputting radius.
- WAP to find the area of a rectangle inputting length and breadth.
- WAP to find Simple Interest



Guess the Output:

1.

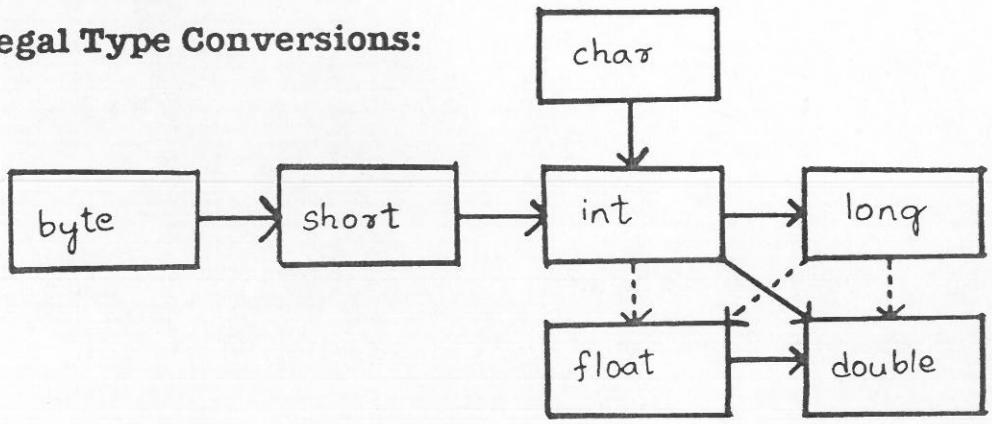
```
public class CharAndBool
{
    public static void main( String [] args )
    {
        char x = 'X';
```

```
        char y = 65;
        int a, b, c ;

        System.out.println( "x = " + x + " and y = " + y );
        System.out.println( "x = " + (int)x + " and y = " + y );
        System.out.println( a = b = c = 1 );

    }
}
```

2.

```
public class TypeCasting
{
    public static void main( String [] args )
    {
        int i = 5;
        float f = i;

        int ii = 6;
        long l = i;

        float ff = 6.0f;
        double d = ff;

        long ll = 5;
        int iii = ll;

        int a = 45;
        byte b = a;

        byte bb = 54;
        char c = bb;

        double zzz = 5.6;
        float yyy = zzz;
    }
}
```

Legal Type Conversions:

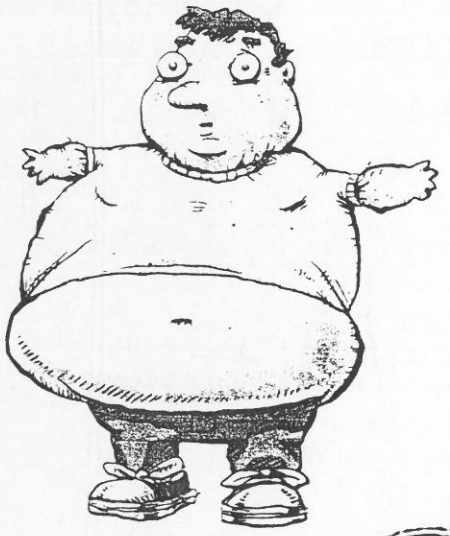


Special Note:

When two values are combined with a binary operator (such as n + f where n is an integer and f is a floating point value), both operands are converted to a common type before the operation is carried out.

- if either of the operands is of type double, the other one will be converted to a double.
- Otherwise, if either of the operands is of type float, the other one will be converted to a float.
- Otherwise, if either of the operands is of type long, the other one will be converted to a long.
- Otherwise, both operands will be converted to an int.

What is "**widening a type**" and "**narrowing a type**" ?

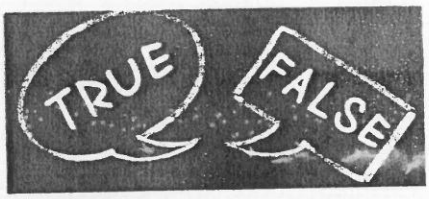



```
byte b = 44;
int  i =  b; // Automatically done, even though we do not do it.

double d = 4.5;
int x = (int)d; // Needs to be done manually. Else a compile time error occurs.
```

Named Constants:

final double PI = 3.14159 (**NOTE**: Use this in your AREA OF CIRCLE program)

State True or False:



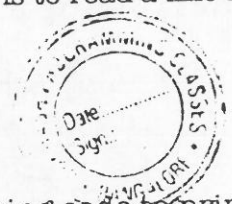
1. **long** to **int** is a legal type conversion :
2. In an expression if either of the operand is **short**, the other would be converted to a **short** :
3. The size of a **char** data type is 1 byte:
4. **$subh123** is a valid variable name:
5. It is mandatory to initialise the variables with values before using it:
6. We cannot change the value of a **final** variable:

Fill in the blanks:

1. The size of an **int** is _____ bytes.
2. _____ method is to read a line from the keyboard.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

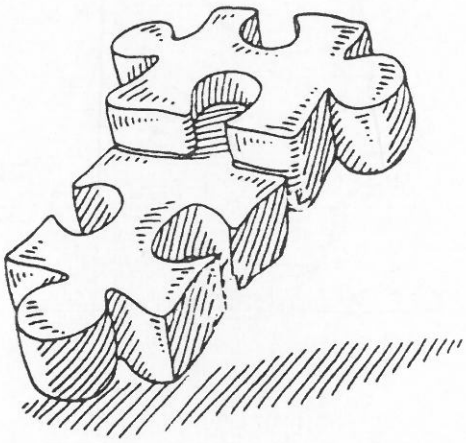
A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___



Try this out:

1. Re-arrange the following code to print the product of 2 numbers.

```
}
public static void main( String [] args )
{
import java.util.*;
System.out.println( "Enter 2 numbers" );
Scanner in = new Scanner( System.in );
num2 = in.nextInt( );
System.out.println( pro );
{
pro = num1 * num2;
int num1, num2, pro;
}
public class Product
num1 = in.nextInt( );
```



Match the following:

To read a line from the keyboard	next( )
To read an integer value from the keyboard	nextLine( )
To read a word from the keyboard	next( ).charAt(0)
To read a character from the keyboard	nextInt( )

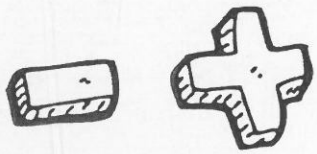
Operators & GUI Basics

Operators:

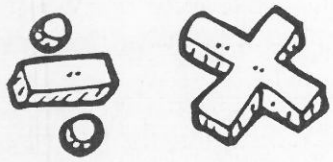
Operators are classified in two ways.

Based on the number of operands used on an operator:

1. Unary
   ++a, -a, --a etc.
2. Binary
   a + b, a % b etc.
3. Ternary
   c = (a > b) ? 1 : 0



Based on the type of operation performed:

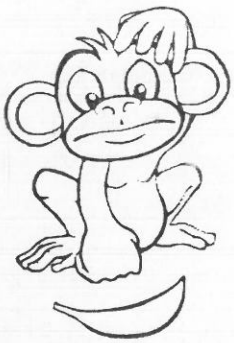


1. Arithmetic
   + - * / %
2. Relational
   < <= > >= == !=
3. Logical
   && || (short circuit operators) ! & | ^
4. Bitwise
   << >> >>> & | ^ ~
5. Compound
   += -= %= >>= etc.
6. Increment/Decrement
   ++a a++ --a a--

Special Note:

There is no explicit operator precedence table in the Java Language Specification and different tables on the web and in text books disagree in some minor ways. (See below code example with testVar to realise). When using the logical AND and OR operators (&& and ||), Java does not evaluate the second operand unless it is necessary to resolve the result(See below code example with testVar to realise).

Guess the Output:



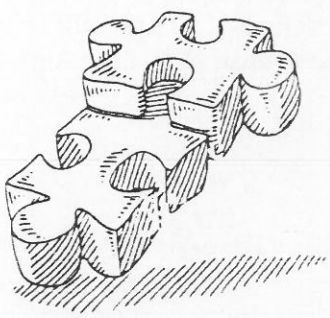
1.

```
public class Operators
{
    public static void main( String [] args )
    {
        int resOne = 1 + 2 * 4 + 5 - 2 + 1 % 2 / 2;
        System.out.println( "ResOne = " + resOne );

        boolean resTwo = 1 < 3 == 4 + 5 <= 6 != 3 < 5 + 1;
        System.out.println( "ResTwo = " + resTwo );

        boolean resThree = 4 + 5 < 6 + 6 && 3 + 1 < 4 || 5 + 1 == 5 + 1;
        System.out.println( "ResThree = " + resThree );

        int testVar = 0, num = 5;
        boolean resFour = testVar != 0 && num / testVar > 10;
        System.out.println( "ResFour = " + resFour );

        boolean boolTest = !true;
        System.out.println( boolTest );

        num = (5 > 6) ? 1 : 0;
        System.out.println(num);

        int val1 = 3;
        val1 += 3.5;
        System.out.println( val1 );

        float val2 = 3.5f;
        val2 += 3;
        System.out.println( val2 );
    }
}
```

Try this out:

- Replace the below statement in the above code and guess what happens:

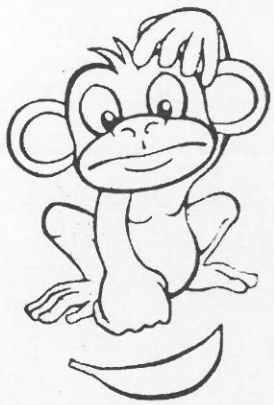
```
boolean resThree = 4 + 5 < 6 + 6 && 3 + 1 || 5 + 1 == 5 + 1;

boolean resFour = testVar != 0 & num / testVar > 10;
```



- WAP to input the time in seconds and output the time in minutes and also print the remaining seconds
- WAP to replace the below statement using bit-wise operators.

  res = (i * 8) + (j / 4);
- Implement GetBits algorithm.
- Implement SetBits algorithm.
- If today is Friday, what will be the day in 50 days ? (**Hint**: Sunday is day 0)

Guess the Output:



1.

```
System.out.println( "326 + 5 is " + 326 + 5 );
System.out.println( "326 + 5.0 is " + 326 + 5.0 );
System.out.println( "10 * (4/8) is " + 10 * (4 / 8));
System.out.println( "10.0 * 2 + 4 is " + 10.0 * 2 + 4);
```

2.

```
double x = 1.0, y = 5.0;
double z = x - - + ( ++y );
x = ? y = ? z = ?
```

Unicode and ASCII Code:

- Mapping a character to its binary representation is called encoding.
- Java uses 'Unicode' encoding scheme to represent world's diverse characters.
- Java supports 1,112,064 characters.
- In a 'char' we can represent characters from '\u0000' to '\uFFFF'.
- Unicode also includes ASCII code with '\u0000' to '\u007F' correspoding to the 128 ASCII characters.

```
public class UnicodeTest
{
    public static void main( String [] args )
    {
        char ch = 'A';
        System.out.println( ch );

        char ch0 = '\u0041';
```

```
        System.out.println( ch0 );

        for( char kan = '\u0C80'; kan <= '\u0CFF'; kan++ )
        {
            System.out.print( kan + " " );
        }
        System.out.println( "\n" );

        char china1 = '\u6B22';
        char china2 = '\u8FCE';

        System.out.print( china1 );
        System.out.print( china2 );

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

Output:

```
A
A
ಂ ಃ ಅ ಆ ಇ ಈ ಉ ಊ ಋ ಌ ಎ ಏ ಐ ಒ ಓ ಔ ಕ ಖ ಗ ಘ ಙ ಚ ಛ ಜ ಝ ಞ ಟ ಠ
ಡ ಢ ಣ ತ ಥ ದ ಧ ನ ಪ ಫ ಬ ಭ ಮ ಯ ರ ಱ ಲ ಳ ವ ಶ ಷ ಸ ಹ ಽ ಾ ಿ ೀ ು
ೂ ೃ ೆ ೇ ೈ ೊ ೋ ೌ ್ ೕ ೖ ೞ ೠ ೡ ೦ ೧ ೨ ೩ ೪ ೫ ೬ ೭
೮ ೯ ೱ ೲ ೳ
欢迎
```

Exponent Operations:

Class Static method

```
System.out.println( Math.pow(2, 3) ); // Displays 8.0
```

The String of Java

```
String name = "Subhash Programming Classes"
System.out.println( name );
```

- WAP to print statements to find the ASCII code for '2', '3', 'a' and 'Z'. Use print statements to find out the character for the decimal codes 50, 65, 66, 99 and 107. Use print statements to find out the character for the hexadecimal code 4B, 54, 57, AD and 6C.
- WAP to print **Hello "Java" world.**

Evaluate:

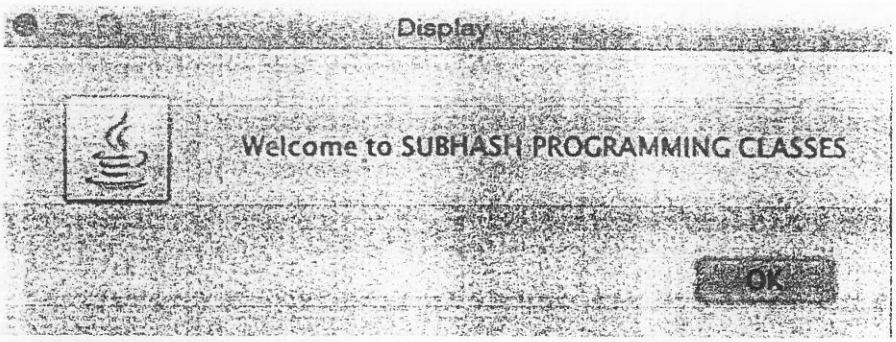
int j = '5' + '6' * ('4' - 50) + 'b' / 2;
What is j = ?

Let us have fun with GUI !!!

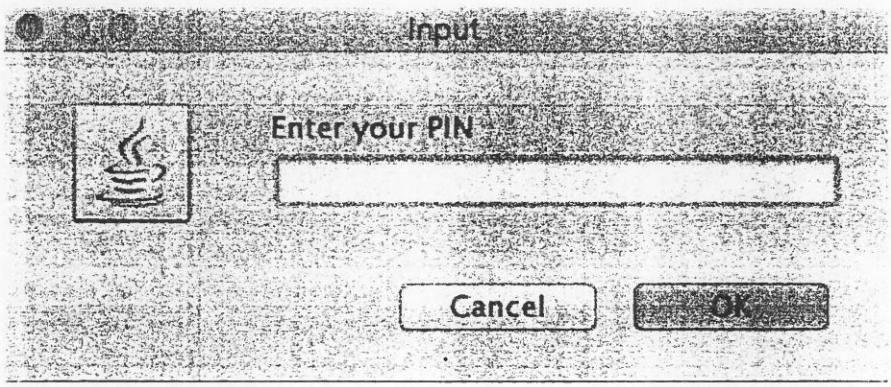
(Do not worry if you are unable to understand few terminologies. Everything would be discussed in detail later. Now just enjoy your first GUI !)


```
import javax.swing.JOptionPane;

public class MyFirstGui
{
    public static void main( String [] args )
    {
        JOptionPane.showMessageDialog( null, "Welcome to SUBHASH
PROGRAMMING CLASSES" );
    }
}
```


```
import javax.swing.JOptionPane;
public class MyFirstGui
{
    public static void main( String [] args )
    {
        JOptionPane.showMessageDialog( null, "Welcome to SUBHASH PRO-
GRAMMING CLASSES", "Display", JOptionPane.INFORMATION_MESSAGE );
    }
}
```



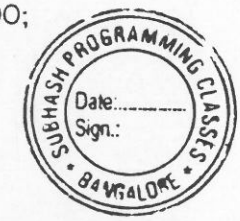

```
import javax.swing.JOptionPane;

public class MySecondGui
{
    public static void main( String [] args )
    {
        JOptionPane.showInputDialog( null, "Enter your PIN" );
    }
}
```

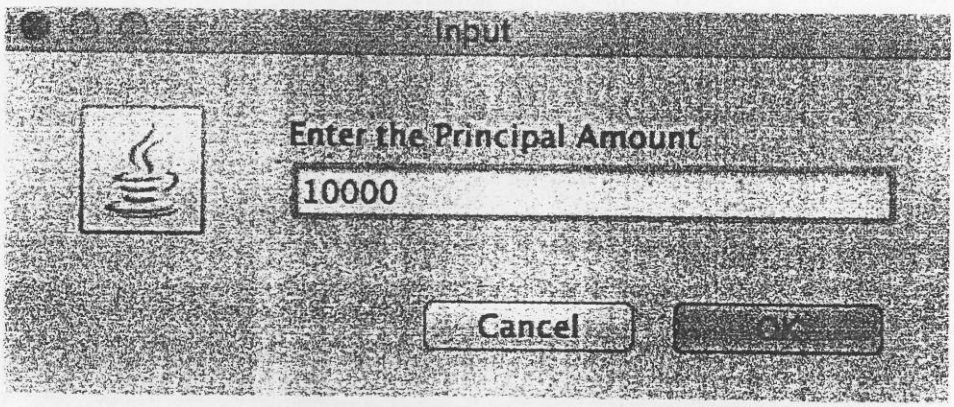


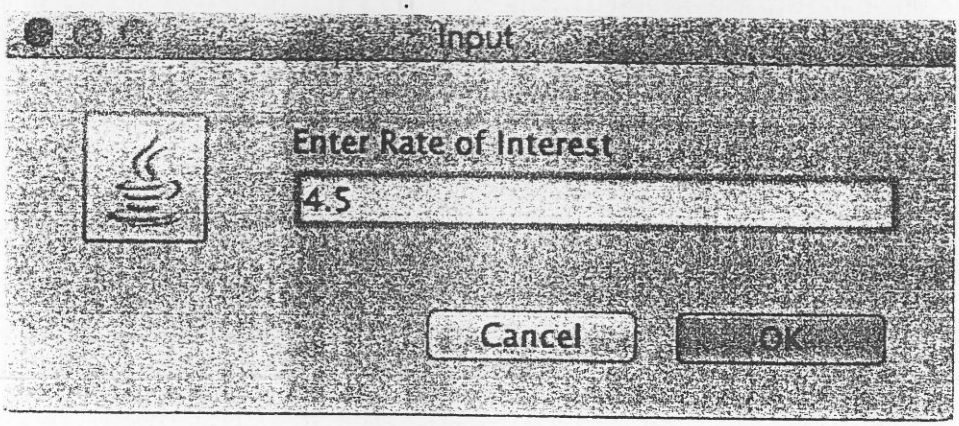
Calculating Simple Interest through GUI !!!

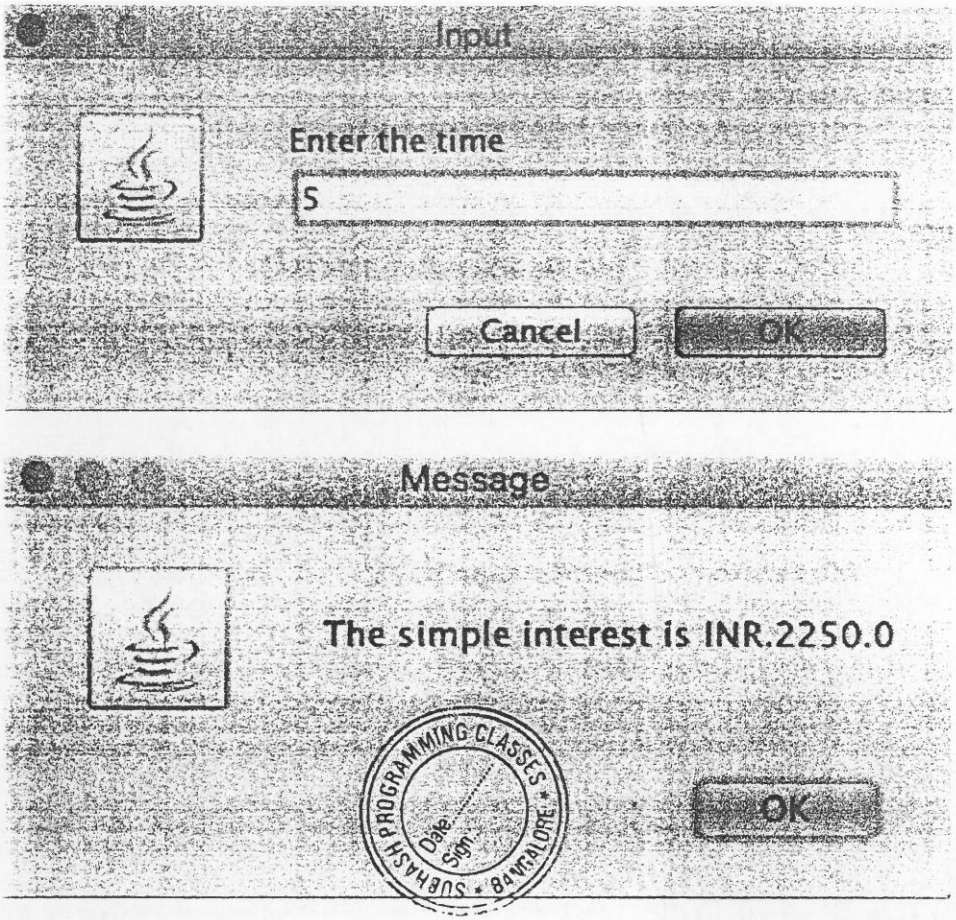
```
import javax.swing.JOptionPane;

public class SimpleInterest
{
    public static void main( String [] args )
    {
        int principalAmount, time;
        double rateOfInterest;
        double si;
```

```
        String paInput = JOptionPane.showInputDialog( null, "Enter the
Principal Amount" );
        principalAmount = Integer.parseInt( paInput );

        String roi = JOptionPane.showInputDialog( null, "Enter Rate of In-
terest" );
        rateOfInterest = Double.parseDouble(roi);

        String t = JOptionPane.showInputDialog( null, "Enter the time" );
        time = Integer.parseInt(t);

        si = ( principalAmount * rateOfInterest * time ) / 100;

        String Output = "The simple interest is INR." + si ;

        JOptionPane.showMessageDialog( null, Output );

    }
}
```

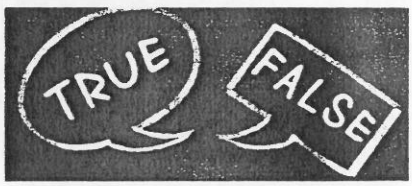








State True or False:



1. &, | are known as short circuit operators :
2. **5 && true** results in true :
3. **true ^ true** results in false :
4. The Unicode also represents ASCII code :
5. **Integer.parseInt( )** is used to convert a String to integer :
6. **showMessageDialog( )** is a static method :
7. << operator can sometimes be used for replacing '*' operator :
8. '**b = a++**' is same as '**b = ++a**' :

Fill in the blanks:

1. Mapping of characters to binary representation is known as _____ .

2. '/' operator gives the _____ and '%' gives the _____ .

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

3. Unicode characters in a 'char' can be represented from _____ to _____ .

4. Among &&& and ||, _____ operator is having the higher priority.

5. There are _____ bitwise operators in Java.

6. _____ operator can be used to mask a bit and _____ operator can be used to set a bit.

Match the following:

&&, ||	Sets the most significant bits to zero while shifting right
&, |, ^	Converts string to integer
Integer.parseInt( )	Short-Circuit Operators
<<	Can be used for division
>>	Can be used for multiplication
>>>	Bitwise operators

Code with Selections

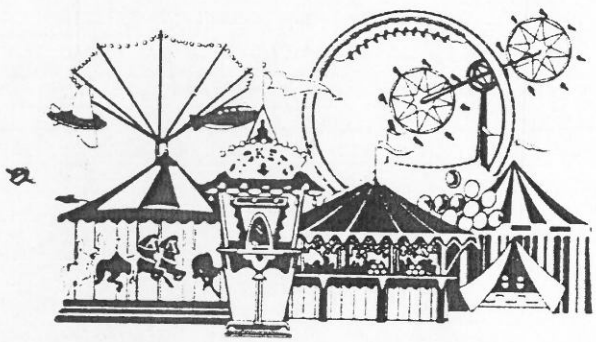
Selections with if, if..else & else..if:

```
if( boolean-expression)
{
    do this;
}
```

```
if( boolean-expression )
{
    do this;
}
else
{
    do this;
}
```


```
if( boolean-expression)
{
        do this;
}
else if( boolean-expression)
{
        do this;
}
else if( boolean-expression)
{
        do this;
}
else if( boolean-expression)
{
        do this;
}
else
{
        do this;
}
```

Selections with switch :



- Switch expressions must yield a value of **char**, **byte**, **short**, **int** or **String** type.


```
i = 3;
switch( i )
{
      case 1 : System.out.println( "Shah Rukh Khan" );
               break;
       case 2: System.out.println( "Deepika Padukone" );
               break;
      case 3: System.out.println( "Subhash.K.U" );
               break;
      case 4: System.out.println( "Katrina Kaif" );
               break;
       case 5: System.out.println( "Akshay Kumar" );
               break;
      default: System.out.println( "Rajinikanth" );
               break;
}
```

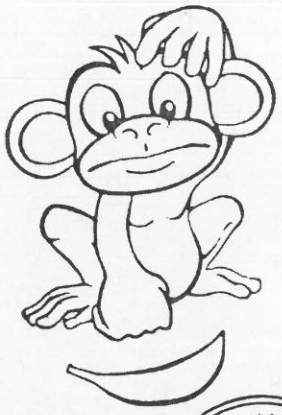
How to generate random numbers ?

double rnum = (**Math.radom**( ) * 10)
- generates a random number between 0.0 and 9.0

int rnum = (int)(**Math.radom**( ) * 10)
- generates a random number between 0 and 9

- WAP to test a school kid whether it knows addition. Don't use if statement.
- WAP to display "Hello Five" if the input number is divisible by 5, "Hello Two" if divisible by 2, "No idea" if none of the above.
- WAP to test a school kid whether it knows Subtraction. Say whether the kid was RIGHT or WRONG.
- WAP to check whether a number is divisible by 2 and 3, by 2 or 3 and by 2 or 3 but not both.
- WAP to determine leap year.
- WAP to win Lottery. If exact 2 digit number matches, Rs.300000 is won. If any of the 2 digit out of 2 digit number matches in any order, Rs.200000 is won. If any 1 digit matches out of 2 digit number, Rs. 100000 is won. Else, you lost. [**Hint**:Use **else..if** ladder]
- WAP to find the birth date.

16	17	18	19	**8**	9	10	11	**4**	5	6	7	**2**	3	6	7	**1**	3	5	7
20	21	22	23	**12**	13	14	15	**12**	13	14	15	**10**	11	14	15	**9**	11	13	15
24	25	26	27	**24**	25	26	27	**20**	21	22	23	**18**	19	22	23	**17**	19	21	23
28	29	30	31	**28**	29	30	31	**28**	29	30	31	**26**	27	30	31	**25**	27	29	31



Guess the output:

1.

```
System.out.println( 'e' < 'f' );
System.out.println( 'a' < 'A' );
System.out.println( 'a' != 'b' );
```

2.

```
boolean b = true;
int i = (int)b;

int k = 25;
boolean b = (boolean)k;
```

3.

```
int a = 6;
if( a > 9 );
{
    System.out.println( "Hello" );
}
```

4.

```
int i = 1, j = 2, k = 3;

if( i > j )
    if( i > k )
        System.out.println("A");
else
    System.out.println( "B");
```

5.

Assume num is 14, 15, and 30. What is the output between these 2 code snippets.

```
if( num % 2 == 0 )
    System.out.println( number + " is even" );
if( num % 5 == 0 )
    System.out.println( number + " is multiple of 5" );
```

```
if( num % 2 == 0 )
      System.out.println( number + " is even" );
else if( num % 5 == 0 )
      System.out.println( number + " is multiple of 5" );
```

6.

```
double x = input.nextDouble( ); // 5.0
double y = input.nextDouble( ); //8.0
double z = input.nextDouble( ); // 9.0

System.out.println( (x > y && y > z ) );
System.out.println( (x > y || y > z ) );
System.out.println( (x < y + z ) );
```

7.

```
System.out.printf( "%8d%8s%8.1f\n", 1234, "Java", 5.63 );
System.out.printf( "%-8d%-8s%-8.1f\n", 1234, "Java", 5.63 );
System.out.printf( "%6b", ( 7 > 2 ));
```

8.

```
int i = 2;
switch(i)
{
      case 0 + i: System.out.println( "Dosa" );
                  break;
      case 2    : System.out.println( "Idli" );
                  break;
      case 3    : System.out.println( "Vada" );
}
```

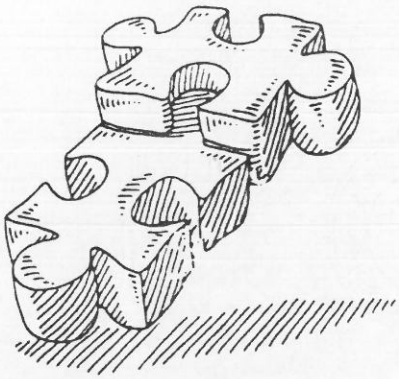
9.

```
int i = 2;
switch(i)
{
      case 2:
      case 1:
      case 3:
      case 9: System.out.println( "Love" );
      default: System.out.println( "Let Love" );
}
```

10.

```
int i = 2;
switch(i)
{
      case 2:
      case 1:
      case 3:
      case 2: System.out.println( "Love" );
```

```
        default: System.out.println( "Let Love" );
    }
```



Try this out:

1.Rewrite the following statement using a Boolean expression:

```
if( count % 10 == 0 )
        newLine = true;
else
        newLine = false;
```

2.How do you generate a random integer number 'i' such that 60 <= i <= 150?

3. Rewrite the following **if** statements using the conditional operator.

```
if( num % 2 == 0 )
        System.out.println( "even" );
else
        System.out.println( "odd" );
```

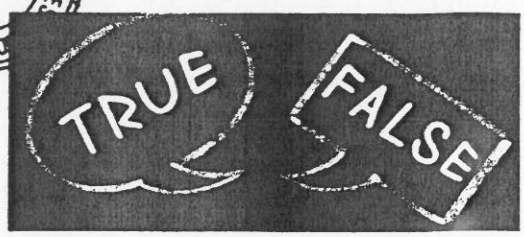
4. Rewrite the following **conditional operator** using **if-else** statements.

```
tax = ( income <=50000 ) ? income * 0.1 : income * 0.12 + 1000;
```

5. How is the following statement evaluated ?

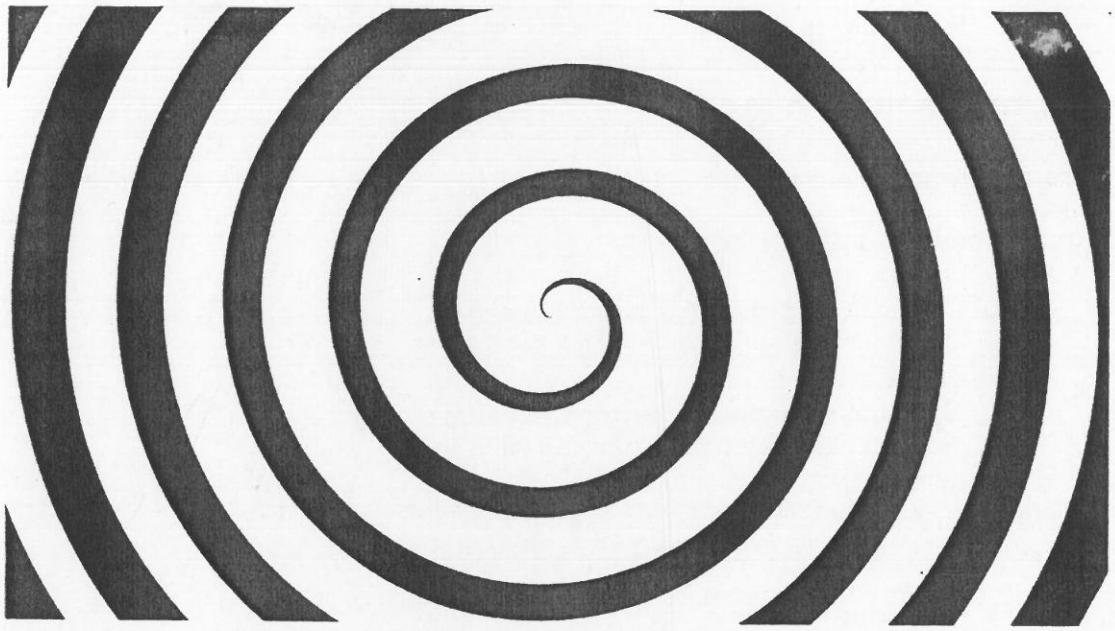
```
a = b += c = 5 ?
```

State True or False:



1. '**float**' expression can be used in a switch statement :
2. Any non-zero value is considered as **true** in java :
3. It is mandatory to put a single statement associated with an if statement within braces :
4. '**break**' statement is mandatory in a '**switch**' statement :
5. If we enter a double value to a nextInt( ) method, the fractional part is truncated :

Code with Loops



1.

```
for( i = 0; i < 10; i++ )
{
    System.out.println( "Subhash Programming Classes" );
}
```

2.

```
i = 0;
while( i < 10 )
{
    System.out.println( "Subhash Programming Classes" );
    i++;
}
```

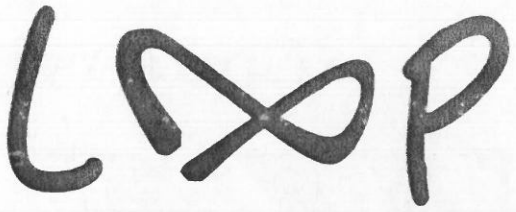
3.

```
i = 0;
do
{
    System.out.println( "Subhash Programming Classes" );
    i++;
}while( i < 10 );
```

Infinite Loops:

```
while(true)
{
    //code
}
```

```
for( ; ; )
{
    //code
}
```



Few sample source codes:

1. To find the sum of n numbers.

```
public class SumOfNumbers
{
    public static void main( String [] args )
    {
        int sum = 0, n;
        System.out.println( "Enter the value of n" );
        Scanner in = new Scanner( System.in );
        n = in.nextInt( );

        for( int i = 0; i <= n; i++ )
        {
            sum = sum + i;
        }
        System.out.println( "Sum of " + n + " numbers is " + sum );
    }
}
```

Sample Run:

```
$ java SumOfNumbers
Enter the value of n
5
Sum of 5 numbers is 15
$
```

2. To repeat the addition quiz.

```
public class AdditionQuiz
{
    public static void main( String [] args )
    {
        int num1 = (int)(Math.random() * 10);
        int num2 = (int)(Math.random() * 10);

        System.out.println( "What is the sum of " + num1 + " and " +
        num2 + " ? " );
```

```
        Scanner in = new Scanner( System.in );
        int Answer = in.nextInt( );

        while( (num1 + num2) != Answer )
        {
            System.out.println( "Wrong Answer. Try Again" );
            System.out.println( "What is the sum of " + num1 + " and " +
             num2 + " ? " );
            Answer = in.nextInt( );
        }
        System.out.println( "Yes. You are right ! " );
    }
}
```

Sample Run:

```
$ java additionquiz
What is the sum of 5 and 1 ?
6
Yes. You are right !

$ java additionquiz
What is the sum of 8 and 2 ?
78
Wrong Answer. Try Again
What is the sum of 8 and 2
```

3. To print multiplication table from 1 to 10. (Demonstrates nested loop)

```
public class multiplication
{
    public static void main( String [] args )
    {
        int i, j;
        System.out.print( "    " );
        for( i = 1; i <= 10; i++ )
        {
            System.out.printf( "%6d", i );
        }
        System.out.println( );
        System.out.print( "    " );
System.out.println( "_____" );

        for( i = 1; i <= 10; i++ )
        {
            System.out.printf( "%2d|",i );
            for( j = 1; j <= 10; j++ )
```

```
                {
                    System.out.printf( "%6d", i * j );
                }
                System.out.println( );
                System.out.printf(" |\n" );
                System.out.printf(" |\n" );
            }
        }
    }
```

Sample Run:

$ java Multiplication

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

- WAP to find the number of bits set in a number.
- WAP to find the GCD of 2 numbers.
- WAP to find whether a given number is prime or not.

Shall we continue ?

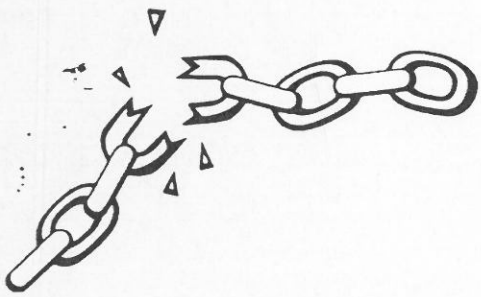
- '**continue**' is used to continue the loops

```
public class Continuing
{
    public static void main( String [] args )
    {
        int i;
        for( i = 0; i < 10; i++ )
        {
            if( i == 1 || i == 3 || i == 6 )
                continue;
            else
                System.out.println( i );
        }
    }
}
```

Shall we break ?

- '**break**' is used to break out of the loops / **switch**

```
public class Breaking
{
    public static void main( String [] args )
    {
        int i;
        for( i = 0; i < 10; i++ )
        {
            if( i == 1 || i == 3 || i == 6 )
                break;
            else
                System.out.println( i );
        }
    }
}
```



Another encounter with 'continue' !

1. '**continue**' without a label:

```
public class Continuing
{
    public static void main( String [] args )
    {
            int i, j;
```

```
            for( i = 0; i < 5; i++ )
            {
                for( j = 0; j <= 5; j++ )
                {
                    if( j == 1 || j == 3 || j == 4 )
                            continue;
                    else
                            System.out.println( j );
                }
            }
            System.out.println( );
        }
    }
```

2. '**continue**' with a label:

```
public class Continuing
{
    public static void main( String [] args )
    {
        int i, j;
        SUBHASH:
        for( i = 0; i < 5; i++ )
        {
            for( j = 0; j <= 5; j++ )
            {
                if( j == 1 || j == 3 || j == 4 )
                        continue SUBHASH; //Labelled continue
                else
                        System.out.println( j );
            }
            System.out.println();
        }
    }
}
```

Another encounter with 'break' !

1. '**break**' without a label:

```
public class Breaking
{
    public static void main( String [] args )
    {
        int i, j;
        for( i = 0; i < 5; i++ )
        {
            for( j = 0; j <= 5; j++ )
```

```
                {
                    if( j == 1 || j == 3 || j == 4 )
                        break;
                    else
                        System.out.println( j );
                }
                System.out.println();
            }
        }
    }
```

2. '**break**' with a label:

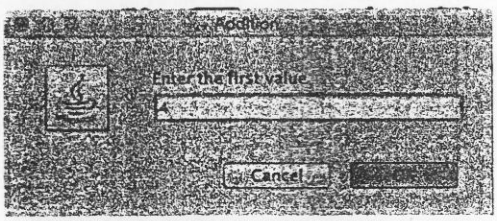
```
public class Breaking
{
    public static void main( String [] args )
    {
        int i, j;
      SUBHASH:
        for( i = 0; i < 5; i++ )
        {
            for( j = 0; j <= 5; j++ )
            {
                if( j == 1 || j == 3 || j == 4 )
                    break SUBHASH; //Labelled break
                else
                    System.out.println( j );
            }
            System.out.println();
        }
    }
}
```

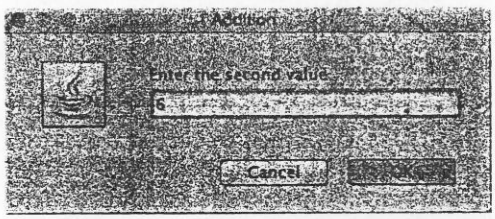
GUI Again with YES or NO option:

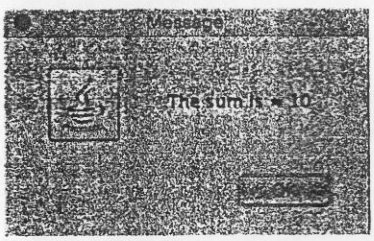
```
import javax.swing.JOptionPane;

public class GuiTest
{
    public static void main( String [] args )
    {
        while(true)
        {
            String value1 = JOptionPane.showInputDialog(null, "Enter the
            first value", "Addition", JOptionPane.QUESTION_MESSAGE );
            int num1 = Integer.parseInt(value1);

            String value2 = JOptionPane.showInputDialog(null, "Enter the
            second value", "Addition", JOptionPane.QUESTION_MESSAGE);
```

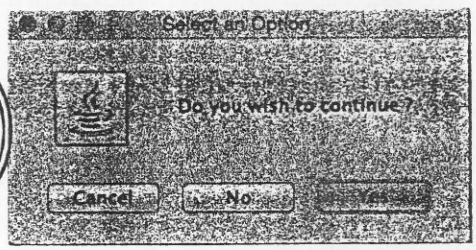
```
                int num2 = Integer.parseInt(value2);

                int res = num1 + num2;

                String sum = "The sum is = " + res;

                JOptionPane.showMessageDialog( null, sum );

                int answer = JOptionPane.showConfirmDialog(null, "Do you wish
                to continue ? " );
                if( answer == JOptionPane.YES_OPTION )
                {
                    continue;
                }
                else
                {
                    break;
                }
            }
        }
}
```

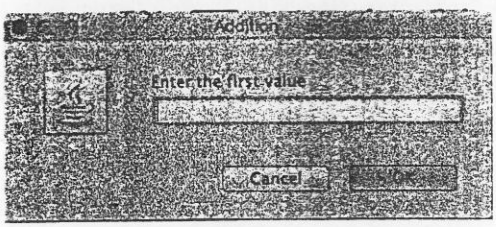
$ java GuiTest



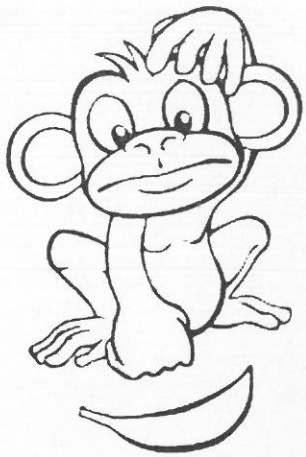








Guess the output:



1.

```
public class test
{
    public static void main( String [] args )
    {
        double item = 1; double sum = 0;
        while( item != 0 )
        {
            sum += item;
            item -= 0.1;
        }
        System.out.println( sum );
    }
}
```

2.

```
public class test
{
    public static void main( String [] args )
    {
        int i = 10;
        while( i >= 1 )
        {
            if( (i--) % 2 == 0 )
                System.out.println( i );
        }
    }
}
```


3.

```
public class test
{
    public static void main( String [] args )
    {
        int i = 10;
        while( i-- >= 1 )
        {
            if( i % 2 == 0 )
                System.out.println( i );
        }
    }
}
```

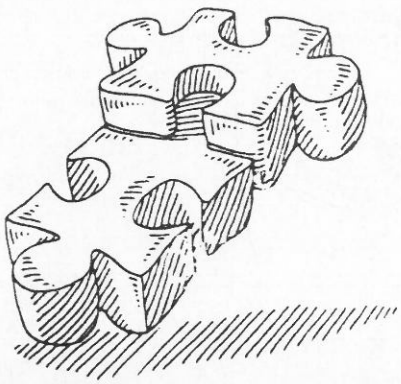
4.

```
public class ShowErrors {
    public static void main( String [] args ) {
        int x = 1;
        while( x <= 10 ) ; {
            System.out.println( x++ );
        }
    }
}
```

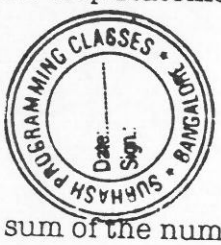
5.

```
Scanner input = new Scanner ( System.in );
int ch, num1, num2;
do
{
    System.out.print( "Enter a number" );
    num1 = input.nextInt( );
    System.out.print( "Enter another number" );
    num2 = input.nextInt( );
    System.out.println( "Their sum is " + (num1 + num2));
    System.out.println( "Do you want to do this again? Press 1 for yes, 0
    for no." );
    ch = input.nextInt( );
}while(ch = 1);
```

Try this out:

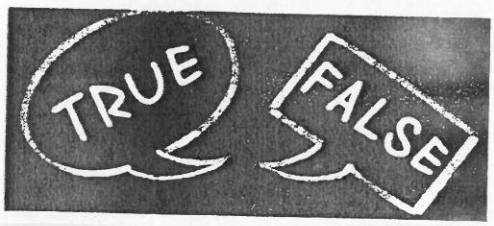


1. Convert the following while loop statement to a for loop.



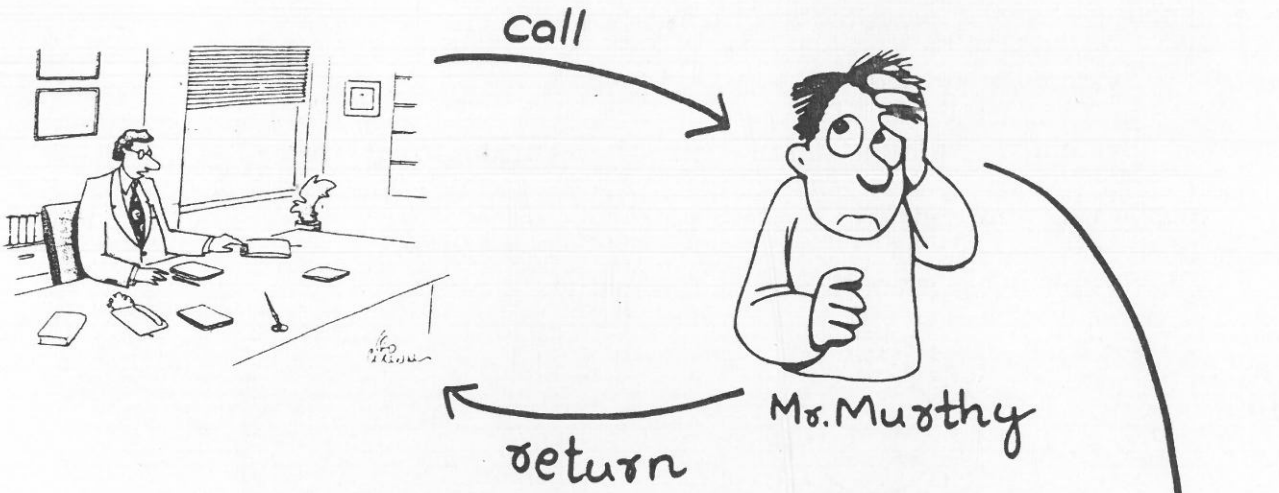

```
int x = 1, sum = 0;
while( x <= 100 )
    sum += x;
    x++;
System.out.print( "The sum of the numbers 1 -
100 is " );
System.out.println( sum );
```

State True or False:



1. '**for**' works faster than '**while**' :
2. In a '**do...while( )**' loop the body executes atleast once :
3. '**continue**' is used within switch statements :
4. '**continue**' and '**labelled continue**', both behave in the same way :
5. '**if**' condition is a looping statement :

Method Invocations



1.

```
public class MethodTest
{
    public static void main( String [] args )
    {
        System.out.println( "My name is Main" );
        murthy( );
    }
    public static void murthy ( )
    {
        System.out.println( "My name is Murthy" );
    }
}
```

2.

```
public class MethodTest
{
    public static void main( String [] args )
    {
        System.out.println( "My name is Main" );
        murthy( 5 );
    }
    public static void murthy ( int num )
    {
        System.out.println( "My name is Murthy" );
        System.out.println( "I received " + num + " from Main " );
    }
}
```

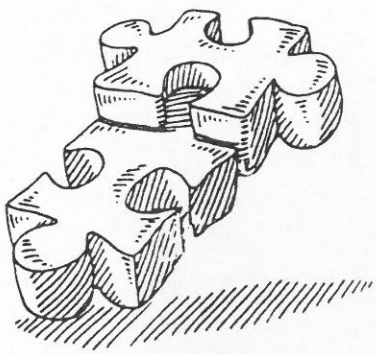
3.

```
public class MethodTest
{
    public static void main( String [] args )
    {
        System.out.println( "My name is Main" );
        int r = murthy( );
        System.out.println( "Main Received " + r + " from Murthy " );
    }
    public static int murthy ( )
    {
        System.out.println( "My name is Murthy" );

        return 8;
    }
}
```

4.

```
public class MethodTest
{
    public static void main( String [] args )
    {
        System.out.println( "My name is Main" );
        int r = murthy( 5 );
        System.out.println( "Main Received " + r + " from Murthy " );

    }
    public static int murthy ( int num )
    {
        System.out.println( "My name is Murthy" );
        System.out.println( "Murthy Received " + num + " from Main " );
        return 8;
    }
}
```

Try this out:



1.

Observe the below code and fill the blanks below it.

```
Line 1 :public class MethodTest
Line 2 :{
Line 3 :    public static int murthy ( int num )
Line 4 :    {
Line 5 :        System.out.println( "My name is Murthy" );
Line 6 :        System.out.println( "Murthy Received " + num + " from Main " );
```

```
Line 7 :        return 8;
Line 8 :    }
Line 9 :    public static void main( String [] args )
Line 10:    {
Line 11:        System.out.println( "My name is Main" );
Line 12:        int r = murthy( 5 );
Line 13:        System.out.println( "Main Received " + r + " from Murthy " );
Line 14:    }

Line 15:}
```

- Who is the calling method ? _____
- Who is the called method ? _____
- What is the return type of the calling method ? _____
- What is the return type of the called method ? _____
- What are the actual arguments of the called method ? _____
- What are the formal arguments of the calling method ? _____
- In which line is the method call ? _____
- What is the name of the variable that is storing the return value ? _____
- Give the starting and ending line numbers of 'murthy' method definition: _____
- Give the starting and ending line numbers of the 'main' method definition: _____

2.

Make a list of all the standard methods that you have learnt till now. Go through the previous chapters to find them. As an example, the first one is done for you. Fill the rest of the blanks. It can be more or less than 10.

Answer:

1. System.out.println( )
2.
3.
4.
5.
6.
7.
8.
9.
10.

Why use Methods ?

- To modularise the code (Organising the code)
- To reuse the code
- To simplify the code

Without Modularising the code:

```
public class SumOfNumbers
{
    public static void main( String [] args )
    {
        int sum = 0;
        for( int i = 1; i <= 5; i++ )
            sum += i;

        System.out.println( "Sum from 1 to 5 is " + sum );

        sum = 0;
        for( int i = 9 ; i <= 45; i++ )
            sum += i;

        System.out.println( "Sum from 9 to 45 is " + sum );

        sum = 0;
        for( int i = 26 ; i <= 39; i++ )
            sum += i;

        System.out.println( "Sum from 35 to 49 is " + sum );

    }
}
```


With Modularising the code:

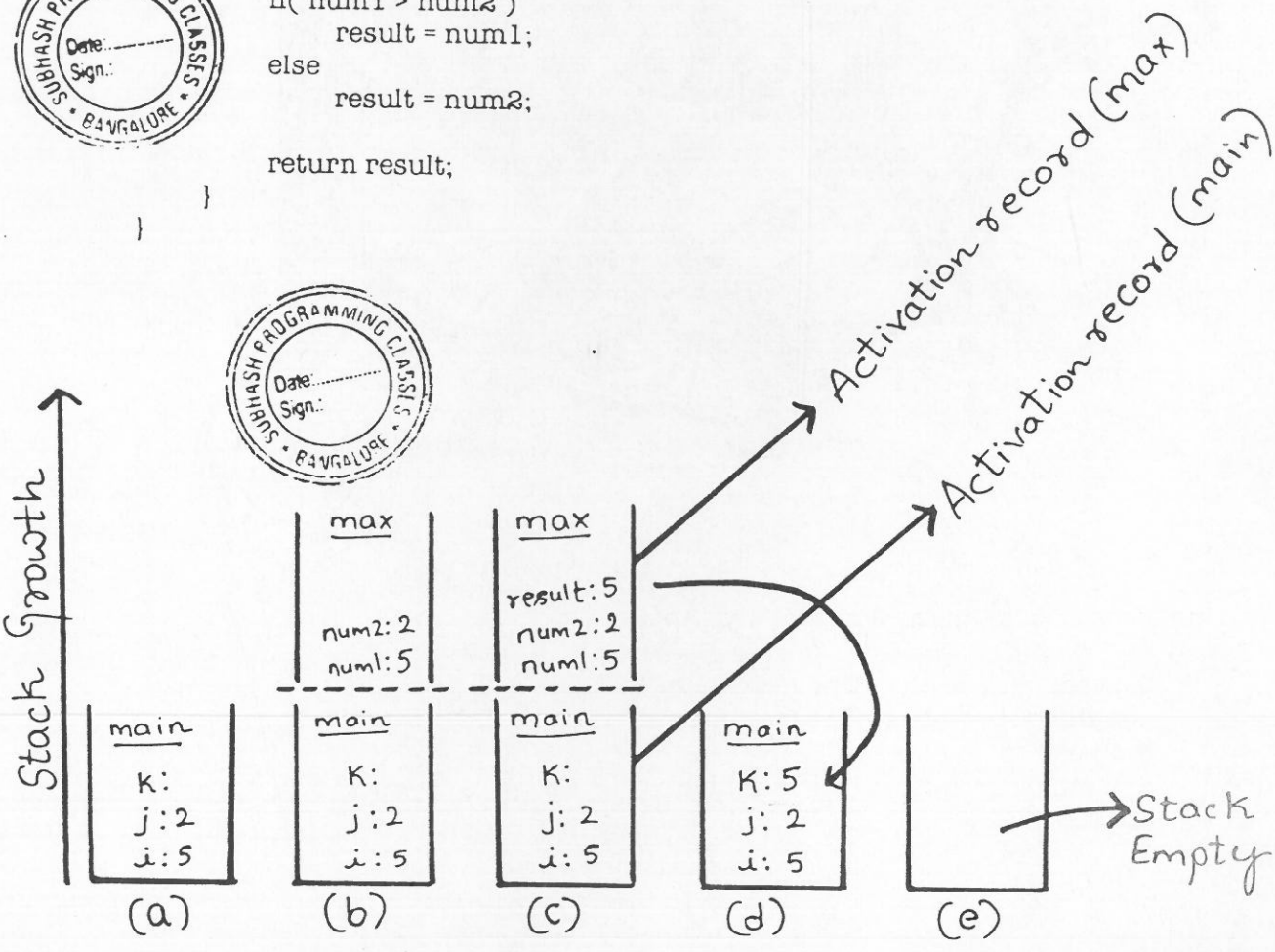
```
public class SumOfNumbers
{
    public static void main( String [] args )
    {
        System.out.println( "Sum from 1 to 10 is " + sumOfN( 1, 5 ));
        System.out.println( "Sum from 20 to 37 is " + sumOfN( 9, 45 ));
        System.out.println( "Sum from 35 to 49 is " + sumOfN( 26, 39 ));
    }

    public static int sumOfN( int range1 , int range2 )
    {
        int sum = 0;
        for( int i = range1; i <= range2; i++ )
            sum += i;

        return sum;
    }
}
```

Stack Growth !

- Stack is where your local variables reside(As part of the activation record). Stack grows when 'methods' are invoked.

A simple program to find the maximum of 2 numbers:

```
public class TestMax
{
    public static void main( String [] args )
    {
        int i = 5;
        int j = 2;
        int k = max( i, j );
        System.out.println( "The maximum of " + i + " and " + j + " is " + k );
    }

    public static int max( int num1, int num2 )
    {
        int result;

        if( num1 > num2 )
            result = num1;
        else
            result = num2;

        return result;
    }
}
```

-WAP using functions to find the smallest of two numbers.

-WAP using functions to convert decimal to hexadecimal.

Method Overloading - Polymorphism !

- Same function name, multiple definitions.
- Depends on the type of parameters, number of parameters and order of parameters.


```
public class TestMax
{
    public static void main( String [] args )
    {
        System.out.println( "The maximum of 3 and 4 is " + max(3, 4) );
        System.out.println( "The maximum of 3 and 4 is " + max(3.0,
        5.4) );
        System.out.println( "The maximum of 3 and 4 is " + max(3.0, 5.4,
        10.14) );
    }
```

```
    public static int max( int num1, int num2 )
    {
        if( num1 > num2 )
            return num1;
        else
            return num2;
    }
    public static double max( double num1, double num2 )
    {
        if( num1 > num2 )
            return num1;
        else
            return num2;
    }

    public static double max( double num1, double num2, double num3 )
    {
        return max( max(num1, num2), num3 );
    }
}
```

Math Class - **The King of Mathematical Functions !**

- Math class contains the methods needed to perform the mathematical functions.

```
public class test
{
    public static void main( String [] args )
    {
        System.out.println( "The value of PI is " + Math.PI );
        System.out.println( "The absolute value of -5 is " + Math.abs(-5));
        System.out.println( "The rounded value of 2.1 is " + Math.round(2.1));
        System.out.println( "The power of 2 and 3 is " + Math.pow(2,3));
        System.out.println( "The max of 2 and 3 is " + Math.max(2,3));
        System.out.println( "The min of 2 and 3 is " + Math.min(2,3));
        System.out.println( "The random number is " + (int)(Math.random( ) * 10));
    }
}
```

Ouput:
The value of PI is 3.141592653589793
The absolute value of -5 is 5
The rounded value of 2.1 is 2
The power of 2 and 3 is 8.0
The max of 2 and 3 is 3
The min of 2 and 3 is 2

Special Note:

All methods in **Math** class are **static** methods.

Strong Recommendation:

www.oracle.com/javase/7/docs/api (For complete documentation on **Math** class)

- WAP using **Math.random**( ) to generate random characters and print on the screen.

Calling methods from another class:

```
class MyMath
{
    public static double abs( double val )
    {
```

```
        if( val < 0 )
            return -(val);
        else
            return val;
    }

    public static int max( int num1, int num2 )
    {
        if( num1 > num2 )
            return num1;
        else
            return num2;
    }

    public static int min( int num1, int num2 )
    {
        if( num1 < num2 )
            return num1;
        else
            return num2;
    }
}

public class MyMathTest
{
    public static void main( String [] args )
    {
        System.out.println( MyMath.abs(-3.5));
        System.out.println( MyMath.max(3, 5));
        System.out.println( MyMath.min(9, 8));
    }
}
```

Output:
3.5
5
8

Guess the Output:

1.

Assume 'n' value is 9

```
public static int sign( int n )
{
    if( n > 0 )
        return 1;
    else if( n == 0 )
        return 0;
```

```
        else if( n < 0 )
            return -1;
    }
```

2.

```
public class SumOfNumbers
{
    public static void main( String [] args )
    {
        fun( );
    }
    public static void fun( )
    {
        return ;
    }
}
```

3.

```
public class Test
{
    public static int method1( int n, int m )
    {
        n += m;
        method2( 7.6 );
    }
    public static method2( char n )
    {
        if( n > 0 )
            return 1;
        else if( n == 0 ) return 0;
        else return -1;
    }
}
```

4.

```
public class SumOfNumbers
{
    public static void main( String [] args )
    {
        System.out.println( "Sum from 1 to 10 is " + sumOfN( 1, 10 ));
        System.out.println( "Sum from 20 to 37 is " + sumOfN( 20, 37 ));
        System.out.println( "Sum from 35 to 49 is " + sumOfN( 35, 49 ));
    }

    public static int sumOfN( int range1 , range2 )
    {
        int sum = 0;
        for( int i = range1; i <= range2; i++ )
            sum += i;
```

```
            return sum;
        }
    }
```

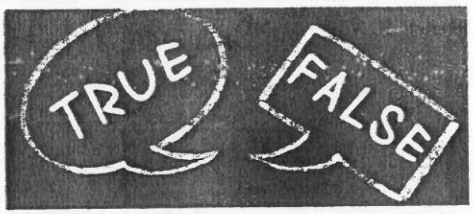
5.

```
public class test
{
    public static void main( String [] args )
    {
        System.out.println( max(1,2));
    }

    public static double max( int num1, double num2 )
    {
        if( num1 > num2 )
            return num1;
        else
            return num2;
    }

    public static double max( double num1, int num2 )
    {
        if( num1 > num2 )
            return num1;
        else
            return num2;
    }
}
```

6.

```
public class MyMathTest
{
    public static void main( String [] args )
    {
        System.out.println( "Subhash Programming Classes" );
        String [] myargs = { "Subhash", " Programming " , " Classes" };
        System.out.println( MyMath.min(9, 8));
        main( myargs );
    }
}
```

State True or False:



1. One method can be defined within another method definition :
2. 'main' method can return an int value :
3. We can return 2 values using return statement :
4. Method overloading does not depend on the return type :
5. Method can be implemented in one class and called from another class :

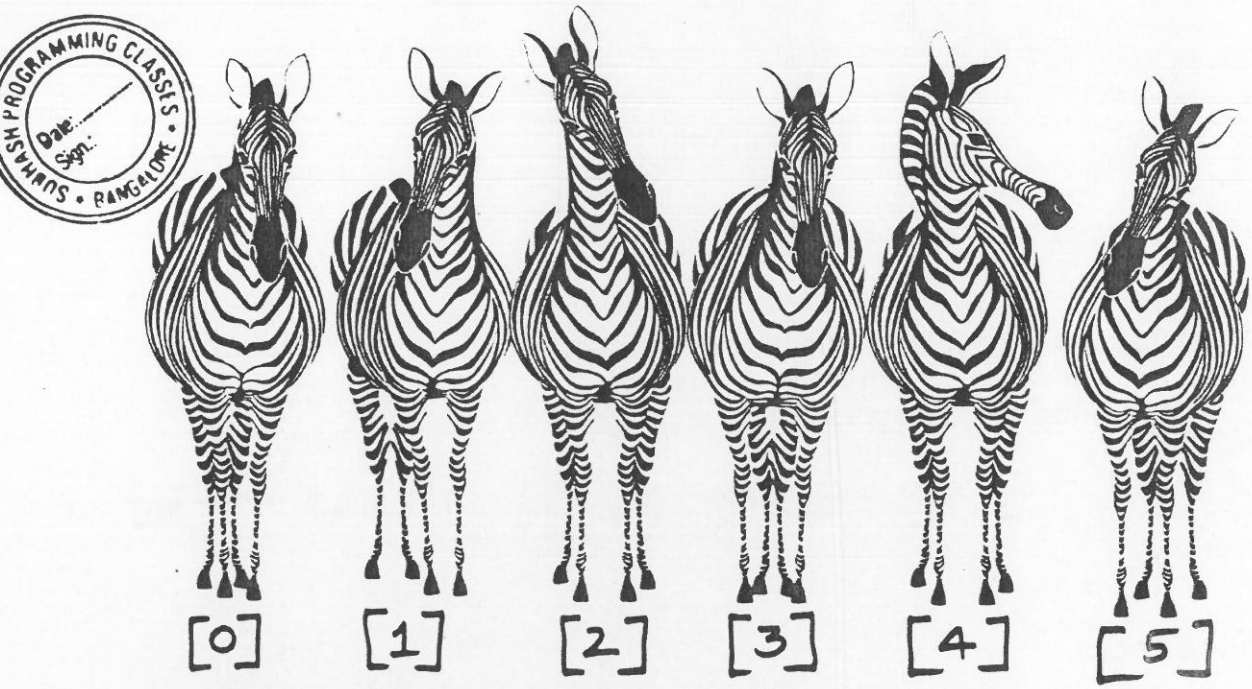
Fill in the blanks:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

1. Each time a method is called a _____ is created on the stack.
2. The concept of same method name but multiple method definitions is called as _____ .
3. To achieve method overloading, the method definitions should differ either in _____ , _____ or _____

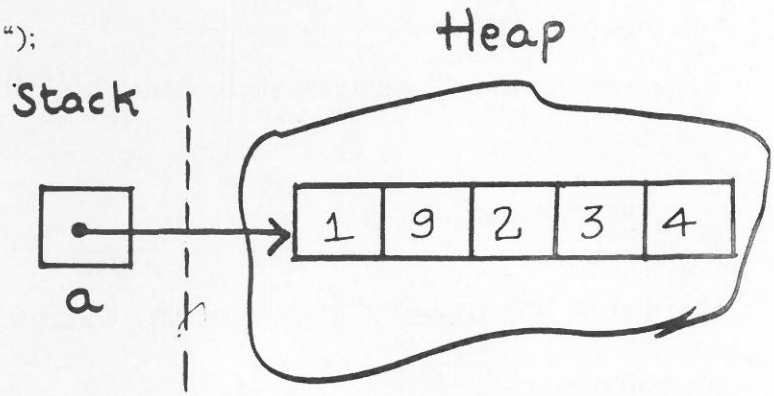
Single-Dimensional Arrays



- Arrays are collections of elements of similar data type.
- Unlike C/C++, an array is an object in java.

```
int [ ] a = { 1, 9, 2, 3, 4 };
for( i = 0; i < a.length; i++ )
{
      System.out.println( a[i] + " ");
}
```

```
int [ ] a = new int [5];
a[0] = 1;
a[1] = 9;
a[2] = 2;
a[3] = 3;
a[4] = 4;
```




```
int [ ] a = new int [5];
for( i = 0; i < a.length; i++ )
{
      a[i] = in.nextInt( );
}
for( i = 0; i < a.length; i++ )
{
      System.out.println( a[i] + " ");
```

}

Other ways:

1.

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

2.

```
int i [ ] = new int [ ] { 3, 4, 3, 2 };
```

- WAP to find the sum of 'n' elements stored in an array.
- WAP to find the largest element in an array.
- WAP to find the smallest index of largest element in an array.
- WAP to shift the elements in an array.
- WAP to find the the number of occurrences of characters in a string.

for-each loop:


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

System.out.println( "Enter 5 elements" );

for( int i = 0; i < a.length; i++ )
{
    a[i] = in.nextInt( );
}
System.out.println( "The entered elements are" );

for( int val : a )
{
    System.out.println( val );
}
```

Variable Number of arguments - VARARGS:

Example - 1:

```
public class VarArgsTest
{
    public static void main( String [] args )
    {
        func( 1 );
```

```
        func(1, 2, 3);
        func(3, 1);
        func( );
    }

    public static void func( int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Contents:
" );

        for( int x : v )
            System.out.println( x + " " );

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

Output:

```
Number of arguments: 1, Contents:
1

Number of arguments: 3, Contents:
1
2
3

Number of arguments: 2, Contents:
3
1
Number of arguments: 0, Contents:
```

Example - 2:

```
public class OLVarArgsTest
{
    public static void main( String [] args )
    {
        func( 1, 2, 3 );
        func( "Testing", 10, 20 );
        func( true, false, false );
    }

    public static void func( int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Contents: " );

        for( int x : v )
            System.out.println( x + " " );
```

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

    public static void func( String msg, int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Contents: " );

        System.out.println( msg );

        for( int x : v )
            System.out.println( x + " " );

    }

    public static void func( boolean ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Contents: " );

        for( boolean x : v )
            System.out.println( x + " " );

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

Output:

```
$ java OLTest
Number of arguments: 3, Contents:
1
2
3

Number of arguments: 2, Contents:
Testing
10
20
Number of arguments: 3, Contents:
true
false
false

$
```

The standard 'Arrays' class:

```
import java.util.*;
public class CommonArrayOperations
```

```
{
    public static void main( String [] args )
    {
        int [] arr = { 23, 45, 11, 54, 89, 32 };
        int i;

        System.out.println( "Original Array" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.print( arr[i] + " " );
        }
        System.out.println( );

        Arrays.sort( arr );
        System.out.println( "\nSorted Array in ascending order" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.print( arr[i] + " ");
        }
        System.out.println( );

        System.out.println( );

        int index = Arrays.binarySearch( arr, 54 );
        System.out.println( "\nElement 54 found at " + index );


        int [] newarr = new int [6];
        newarr = Arrays.copyOf(arr, arr.length );
        System.out.println( "New array contents" );
        for( i = 0; i < newarr.length; i++ )
        {
            System.out.println( newarr[i] + " " );
        }

        Arrays.fill( arr, 0 );
        System.out.println( "\nCleard Array" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.println( arr[i] + " " );
        }
        System.out.println( );
    }
}
```

```
            for( int x : v )
                System.out.println( x + " " );

            System.out.println( );
        }
        public static void func( int n, int ... v )
        {
            System.out.println( "Number of arguments: " + v.length + ", Contents: " );

            for( int x : v )
                System.out.println( x + " " );

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

8.

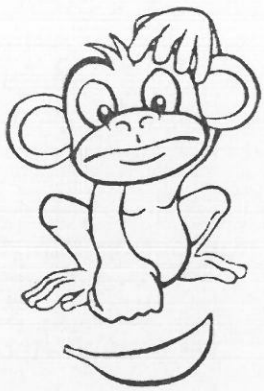
```
public class ArrayPassingTest
{
    public static void main( String [] args )
    {
        int [] marks = { 5, 6, 7, 5, 7, 7, 9 };
        int i;

        for( i = 0; i <= 6; i++ )
        {
            Modify(marks[i]);
        }

        for( i = 0; i <= 6; i++ )
        {
            System.out.println( marks[i]);
        }
    }
    static void Modify( int m )
    {
        m = m * 2;
    }
}
```

9.

```
public class ArrayPassingReferenceTest
{
    public static void main( String [] args )
    {
        int [] marks = { 5, 6, 7, 5, 7, 7, 9 };
```

```
{
    public static void main( String [] args )
    {
        int [] arr = { 23, 45, 11, 54, 89, 32 };
        int i;

        System.out.println( "Original Array" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.print( arr[i] + " " );
        }
        System.out.println( );

        Arrays.sort( arr );
        System.out.println( "\nSorted Array in ascending order" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.print( arr[i] + " ");
        }
        System.out.println( );

        System.out.println( );

        int index = Arrays.binarySearch( arr, 54 );
        System.out.println( "\nElement 54 found at " + index );


        int [] newarr = new int [6];
        newarr = Arrays.copyOf(arr, arr.length );
        System.out.println( "New array contents" );
        for( i = 0; i < newarr.length; i++ )
        {
            System.out.println( newarr[i] + " " );
        }

        Arrays.fill( arr, 0 );
        System.out.println( "\nCleard Array" );
        for( i = 0; i < arr.length; i++ )
        {
            System.out.println( arr[i] + " " );
        }
        System.out.println( );
    }
}
```

Guess the output:



1.

```
double [] v;
v = { 1.2, 2.3, 3.4, 5.1 };
```

2.

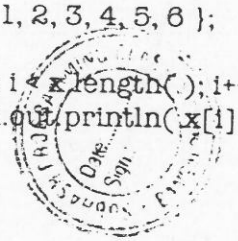
```
char [ ] c = { 'S', 'U', 'B', 'H', 'A', 'S', 'H' };
System.out.println(c);
```

3.

```
for( int i = 0; i <= a.length; i++ )
{
    System.out.println( a[i] + " " );
}
```

4.

```
public class Test
{
    public static void main( String [] args )
    {
        int [6] x = { 1, 2, 3, 4, 5, 6 };

        for( int i = 0; i < x.length( ); i++ )
            System.out.println( x[i] );
    }
}
```



5.

```
public class Test
{
    public static void main( String [] args )
    {
        int [6] x = { 1, 2, 3, 4, 5, 6 };

        for( int i = 0; i < x.length( ); i++ )
            System.out.println( x[i] );
    }
}
```

6.

```
public class test
{
    public static void main( String [] args )
    {
        double d [ ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
        for( int i = 0; i < d.length; i++ )
```

```
                d[i] = d[i] + 1.0;

            for( int i = d.length - 1; i >= 0; i-- )
                System.out.println( d[i] + " " );
        }
    }
```

6.

```
public class VarArgsTestAmbOne
{
    public static void main( String [] args )
    {
        func(1, 2, 3);
        func(true, false, true);
        func();
    }

    public static void func( int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ",
Contents:" );
        for( int x : v )
            System.out.println( x + " " );

        System.out.println( );
    }
    public static void func( boolean ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Con-
tents: " );

        for( boolean x : v )
            System.out.println( x + " " );

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


7.

```
public class VarArgsTestAmbTwo
{
    public static void main( String [] args )
    {
        func(1);
    }

    public static void func( int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Con-
tents: " );
```

```
        for( int x : v )
            System.out.println( x + " " );

        System.out.println( );
    }
    public static void func( int n, int ... v )
    {
        System.out.println( "Number of arguments: " + v.length + ", Contents: " );

        for( int x : v )
            System.out.println( x + " " );

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

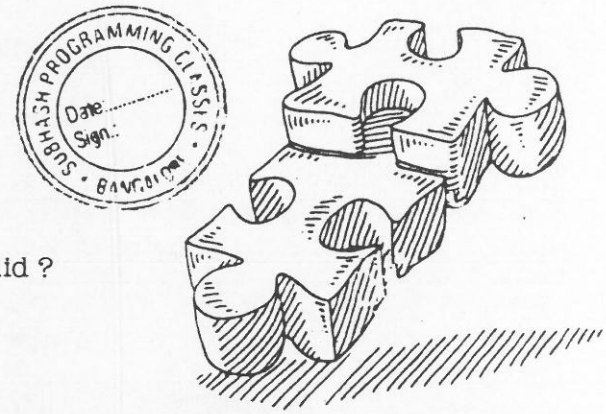
8.

```
public class ArrayPassingTest
{
    public static void main( String [] args )
    {
        int [] marks = { 5, 6, 7, 5, 7, 7, 9 };
        int i;

        for( i = 0; i <= 6; i++ )
        {
            Modify(marks[i]);
        }

        for( i = 0; i <= 6; i++ )
        {
            System.out.println( marks[i]);
        }
    }
    static void Modify( int m )
    {
        m = m * 2;
    }
}
```

9.

```
public class ArrayPassingReferenceTest
{
    public static void main( String [] args )
    {
        int [] marks = { 5, 6, 7, 5, 7, 7, 9 };
```

```
        int i;

        Modify(marks);
        for( i = 0; i < marks.length; i++ )
        {
            System.out.print( marks[i] + " " );
        }
        System.out.println();
    }
    static void Modify( int [] m )
    {
        int i;
        for( i = 0; i < m.length; i++ )
        {
            m[i] = m[i] * 2;
        }
    }
}
```



Try this out:

1. Which of the following statements are valid ?

   a. int i[30];
   b. double d[30] = new double[ ];
   c. char [] r = new char [10];
   d. int i[ ] = new int { 3, 4, 3, 2 };
   e. double d[ ] = { 2, 4, 6 };
   f. float f[ ] = new float [ ];

State True or False:

1. Arrays can have a combination of all types of elements :
2. Accessing beyond the total allocated memory in an array results in a compile time error :
3. Arrays are objects too and they resided on the heap :

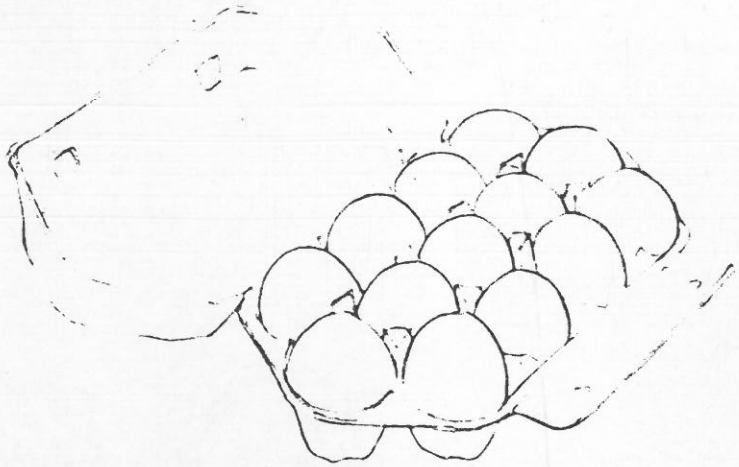
Fill in the blanks:

1. Variable number of arguments for an integers are written as _____ .
2. Arrays are stored on the _____ .

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW___	Y ___

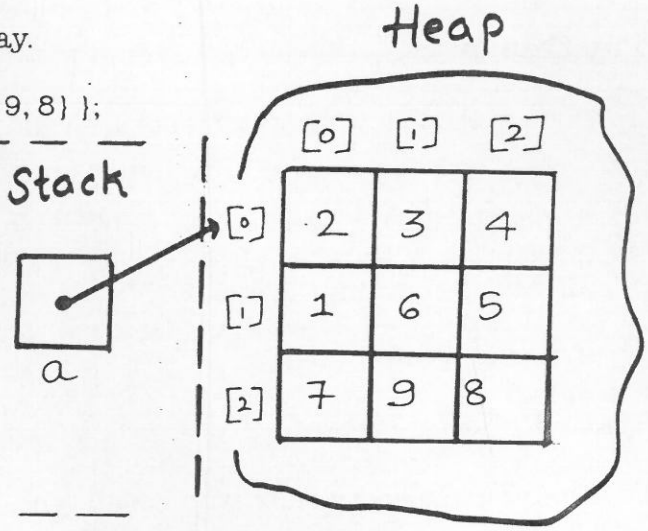
Multi-Dimensional Arrays



- Think of matrix, think of 2-D Array.

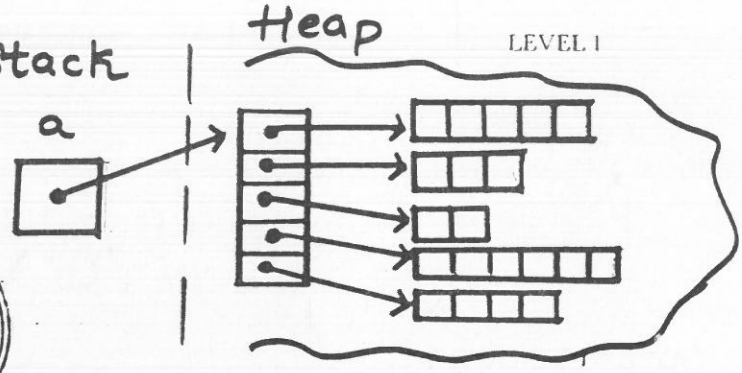
```
int [][] a = { { 2, 3, 4 }, { 1, 6, 5}, {7, 9, 8} };
```

```
int [][] a = new int [3][3];
a[0][0] = 2;
a[0][1] = 3;
a[0][2] = 4;
a[1][0] = 1;
a[1][1] = 6;
a[1][2] = 5;
a[2][0] = 7;
a[2][1] = 9;
a[2][2] = 8;
```


```
int [][] a = new int [3][3];
for( i = 0; i < 3; i++ )
{
      for( j = 0; j < 3; j++ )
      {
            a[i][j] = in.nextInt( );
      }
}
for( i = 0; i < 3; i++ )
{
      for( j = 0; j < 3; j++ )
      {
            System.out.println(a[i][j] + " " );
      }
}
```

```
for( i = 0; i < a.length; i++ )
{
      for( j = 0; j < a[i].length; j++ )
      {
            a[i][j] = in.nextInt( );
      }
}
for ( i = 0; i < a.length; i++ )
{
      for( j = 0; j < a[i].length; j++ )
      {
      System.out.println(a[i][j]+ " " );
      }
}
```



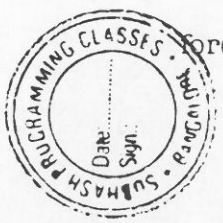
Ragged Arrays:

```
int [][] a = new int [5][];
a[0] = new int [5];
a[1] = new int [3];
a[2] = new int [2];
a[3] = new int [6];
a[4] = new int [4];
```

- WAP to find the sum of all elements in a 2-D array.
- WAP to find the sum of all elements in an individual column of a 2-D array.
- WAP to find the row that has the largest sum in a 2-D array.

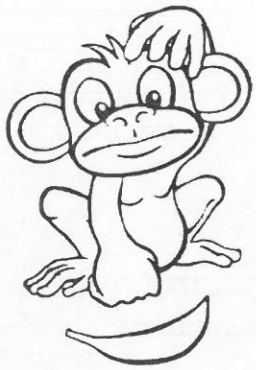
Passing & Returning 2-D Arrays to and from functions:

```
public class test
{
    public static void main( String [] args )
    {
        int [][] m = { { 2, 1, 5 }, { 3, 2, 6 }, { 6, 7, 9}};

        int [][] n = passing( m );

        System.out.println( "I am in main function" );

        for( int i = 0; i < n.length; i++ )
        {
            for( int j = 0; j < n[i].length; j++ )
            {
                System.out.print( n[i][j] + " " );
            }
            System.out.println( );
        }
    }
    public static int [][] passing( int [][] x )
    {
        System.out.println( "I am in Passing function" );

        for( int i = 0; i < x.length; i++ )
        {
            for( int j = 0; j < x[i].length; j++ )
            {
                System.out.print( x[i][j] + " " );
            }
            System.out.println( );
```



```
        }

        int [][]y = { { 4, 4, 4 }, { 5, 5, 5 }, { 6, 6, 6 } };
        return y;
    }
}
```

Guess the output:



1.

```
int [][] array = { {23, 62}, {43, 46}, {51, 36}, {51, 36}};
for( int i = array.length; i>= 0; i- -) {
      for( int j = arr[i].length; j >= 0; j- - )
System.out.print( array[i][j] + " " );
      System.out.println( );

}
```

2.

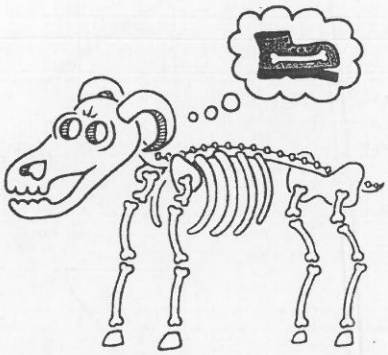
```
int [][] array = { {5, 6}, {8, 5}, {9, 8}};
int product = 1;
for( int i = 0; i < array.length; i++ )
      product *= array[i][0];
System.out.println(product);
```

3.

```
public class Test
{
      public static void main( String [] args )
      {
            int [][] array = { {1, 2, 3, 4}, {5, 6, 7, 8} };
            System.out.println( m1(array)[0] );
            System.out.println( m1(array)[1] );
      }

      public static int [] m1( int [][] m )
      {
            int [] result = new int [2];
            result[0] = m.length;
            result[1] = m[0].length;
            return result;
      }
}
```

Object-Oriented Thinking

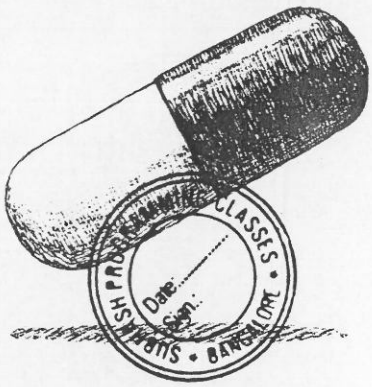
- Classes & Objects



- Close your eyes. Think of any thing. If you can see it clearly, it is an object. Else it is a class.
- Object is something that occupies space, has a state and have a behaviour.

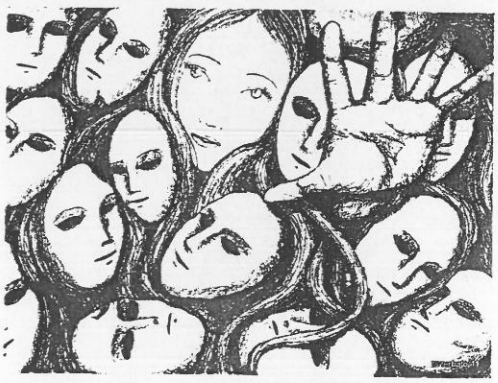
- Encapsulation

- Putting together the state and behaviour together.



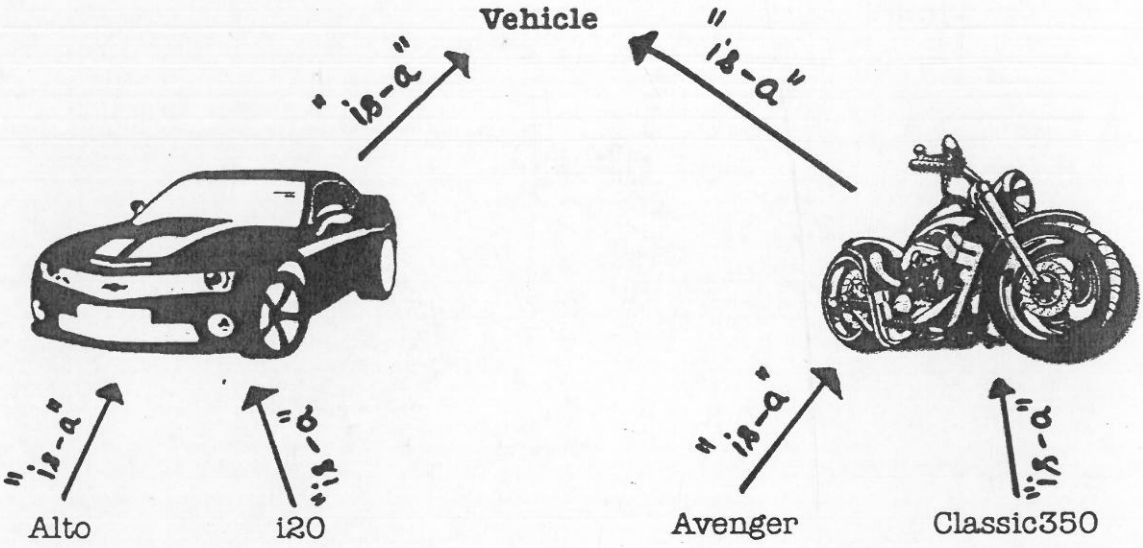
- Polymorphism

- One entity, different forms



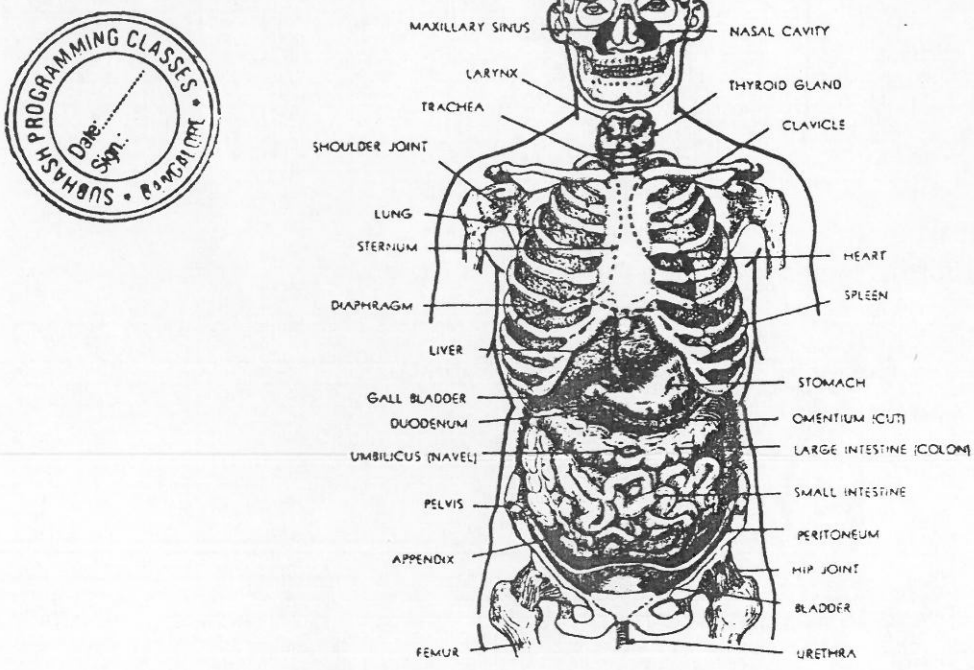
- Inheritance

- That which supports **"is-a"** or **"is-like-a"** relationship

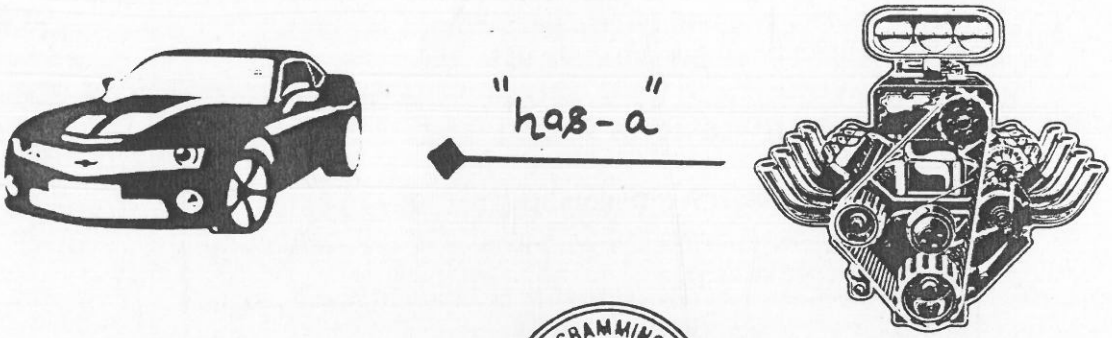


- Abstraction

- Hiding internal details, exposing interface



- **Compostion**
  - That which supports "**has-a**" relationship

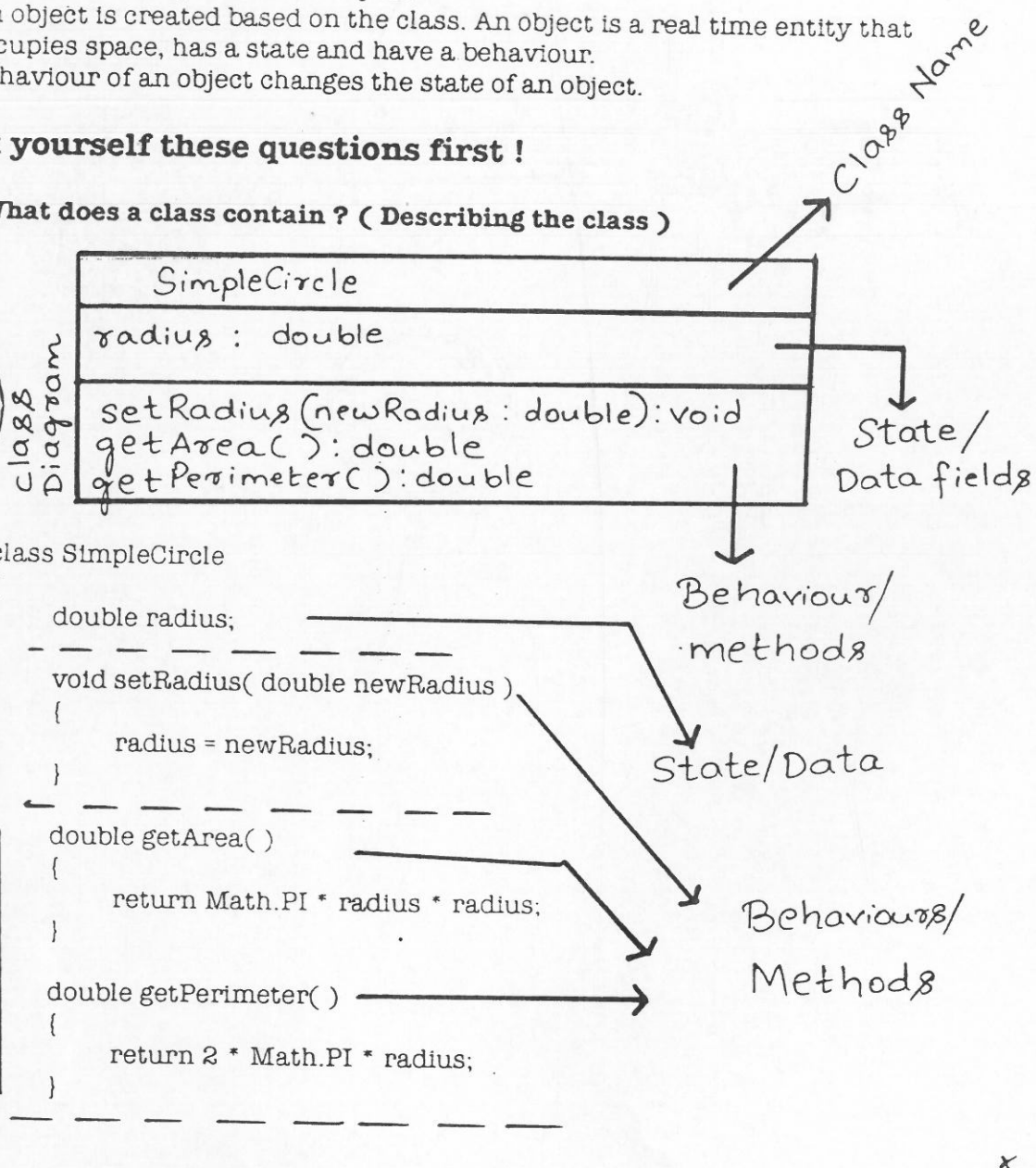


Classes & Objects

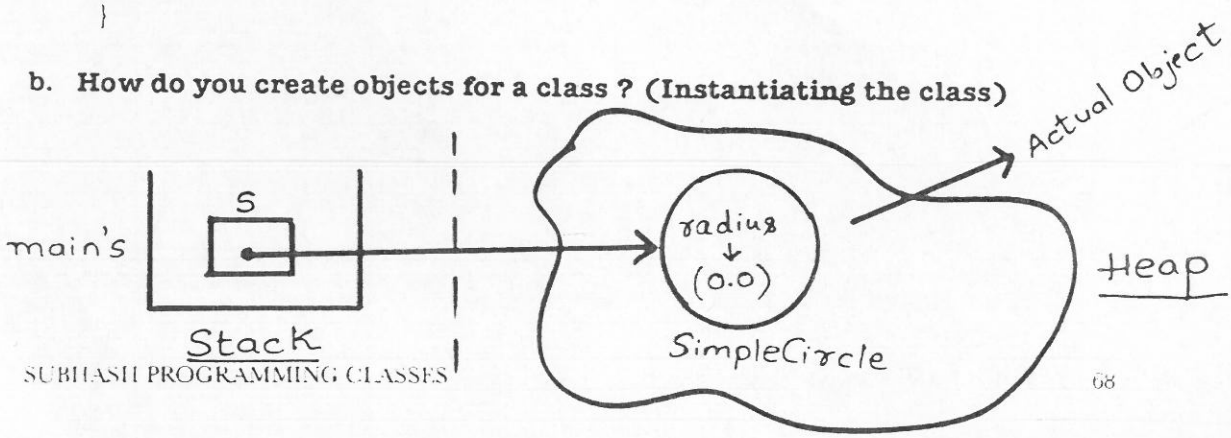
- A class is a blue print for an object.
- An object is created based on the class. An object is a real time entity that occupies space, has a state and have a behaviour.
- Behaviour of an object changes the state of an object.

Ask yourself these questions first !

a. What does a class contain ? (Describing the class)



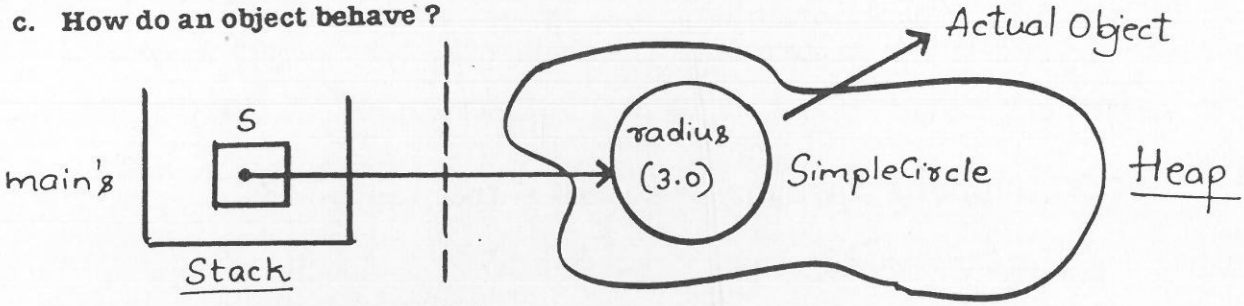
b. How do you create objects for a class ? (Instantiating the class)



```
public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle s = new SimpleCircle( );
    }
}
```

Driver Class or Launcher Class

c. How do an object behave ?




```
public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle c = new SimpleCircle( );

        c.setRadius(3);

        System.out.println( "Area of circle = " + c.getArea( ));
        System.out.println( "Perimeter of circle = " + c.getPerimeter( ));
    }
}
```

The Complete Code !

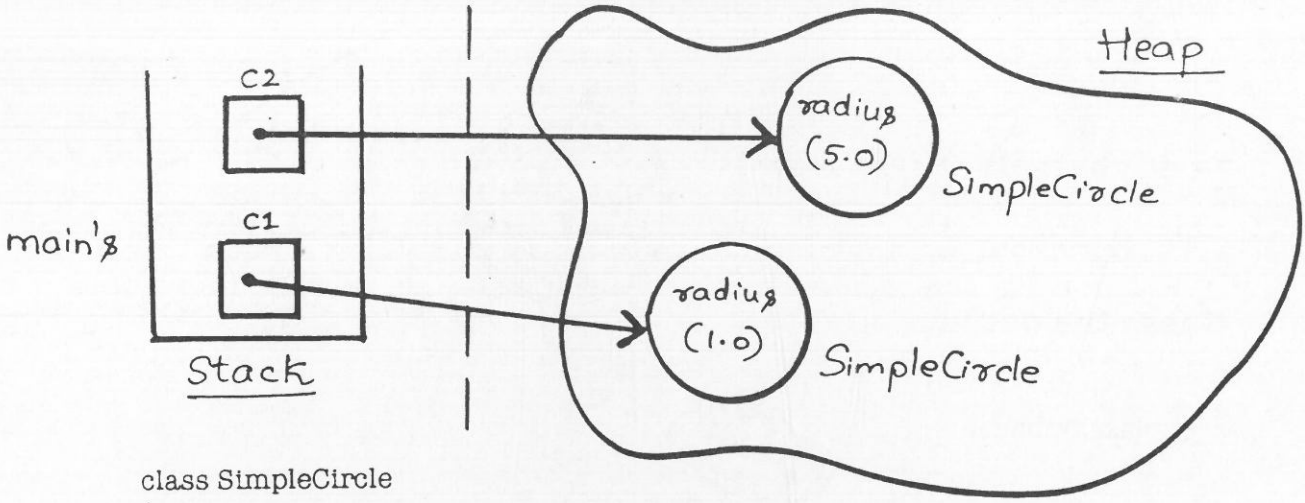
```
class SimpleCircle
{
    double radius;

    void setRadius( double newRadius )
    {
        radius = newRadius;
    }
    double getArea( )
    {
        return Math.PI * radius * radius;
    }
    double getPerimeter( )
    {
        return 2 * Math.PI * radius;
    }
```

```
}

public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle c = new SimpleCircle( );
        c.setRadius(3);
        System.out.println( "Area of circle = " + c.getArea( ));
        System.out.println( "Perimeter of circle = " + c.getPerimeter( ));
    }
}
```

What happens if we forget to call setter method ?

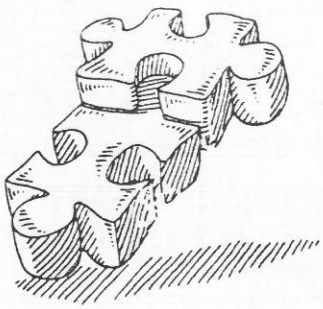
```
class SimpleCircle
{
    double radius;

    SimpleCircle( )
    {
        radius = 1;
    }

    double getArea( )
    {
        return Math.PI * radius * radius;
    }

    double getPerimeter( )
    {
        return 2 * Math.PI * radius;
    }
}

public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle c1 = new SimpleCircle( );
        System.out.println( "Area of circle = " + c.getArea( ));
        System.out.println( "Perimeter of circle = " + c.getPerimeter( ));
    }
}
```

How to create multiple circles with varying radius values ?




```
class SimpleCircle
{
    double radius;

    SimpleCircle( ) // zero-argument constructor
    {
        radius = 1;
    }

    SimpleCircle( double newRadius ) //parameterized constructor
    {
        radius = newRadius;
    }

    double getArea( )
    {
        return Math.PI * radius * radius;
    }

    double getPerimeter( )
    {
        return 2 * Math.PI * radius;
    }
}

public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle c1 = new SimpleCircle( );
        System.out.println( "Area of circle = " + c1.getArea( ));
        System.out.println( "Perimeter of circle = " + c1.getPerimeter( ));

        SimpleCircle c2 = new SimpleCircle( 5 );
```

```
            System.out.println( "Area of circle = " + c2.getArea( ));
            System.out.println( "Perimeter of circle = " + c2.getPerimeter( ));
        }
    }
```



Do it yourself - Right Now !

a. TV (Television)

Guess the output:

1.

```
class Subhash
{
    int a;
    char b;
    float c;
    double d;
    boolean e;
}
public class test
{
    public static void main( String [] args )
    {
        System.out.println(new Subhash().a);
        System.out.println(new Subhash().b);
        System.out.println(new Subhash().c);
        System.out.println(new Subhash().d);
        System.out.println(new Subhash().e);
    }
}
```

2.

```
class Subhash
{
    int a;
}
public class test
{
    public static void main( String [] args )
    {
        int x;
        String y;
        System.out.println(a);
        System.out.println(y);
    }
}
```

3.

```
class Subhash
{
    int a;
}.
public class test
{
    public static void main( String [] args )
    {
        Subhash s = new Subhash;
        System.out.println(s.a);
    }
}
```

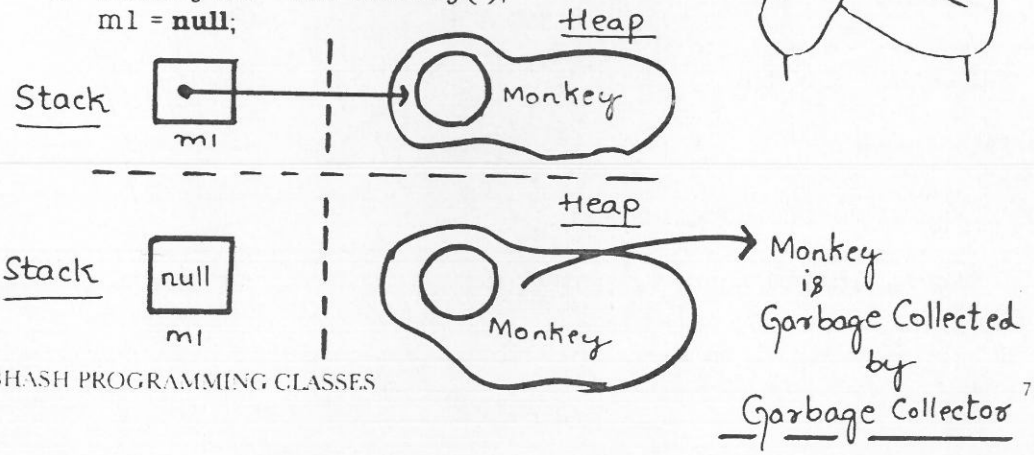
4.

```
class Subhash
{
    private int a;
    public int b;
    int c;
}
public class test
{
    public static void main( String [] args )
    {
            Subhash s = new Subhash( );
            System.out.println( s.a );
            System.out.println( s.b );
            System.out.println( s.c );
    }
}
```

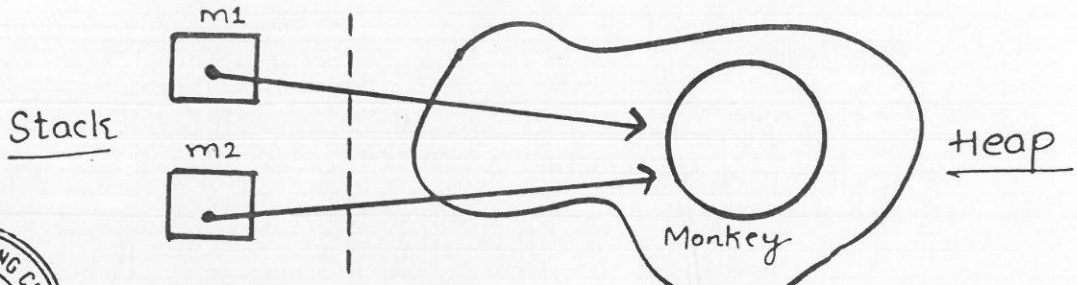
Guess What happens ?

```
class Monkey
{
    int sound;
}
```

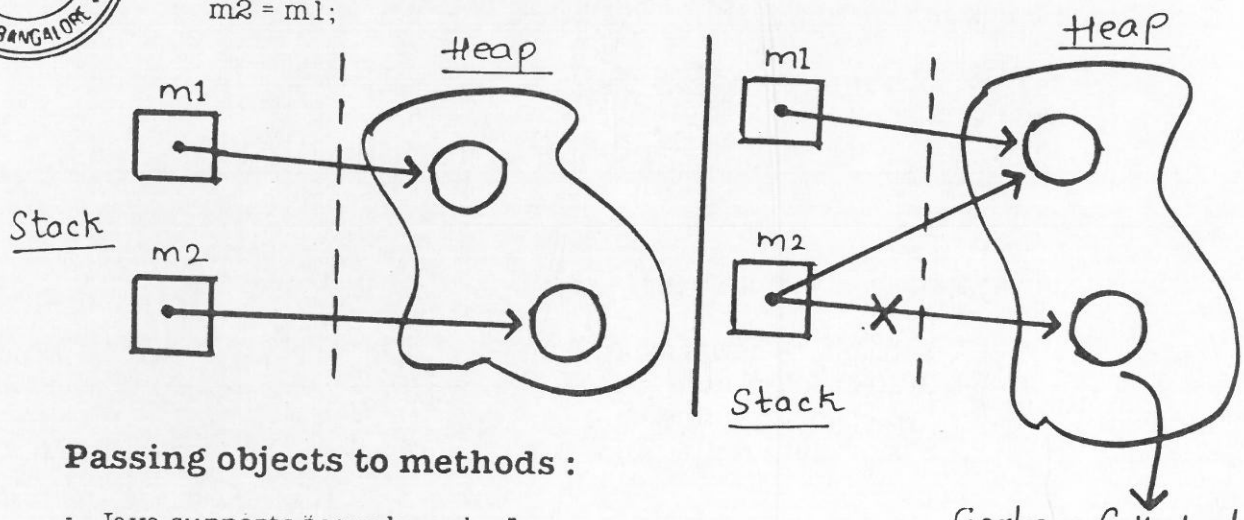
a. Monkey m1 = new Monkey( );
m1 = **null**;



b. Monkey m1 = new Monkey( );
Monkey m2;
m2 = m1;

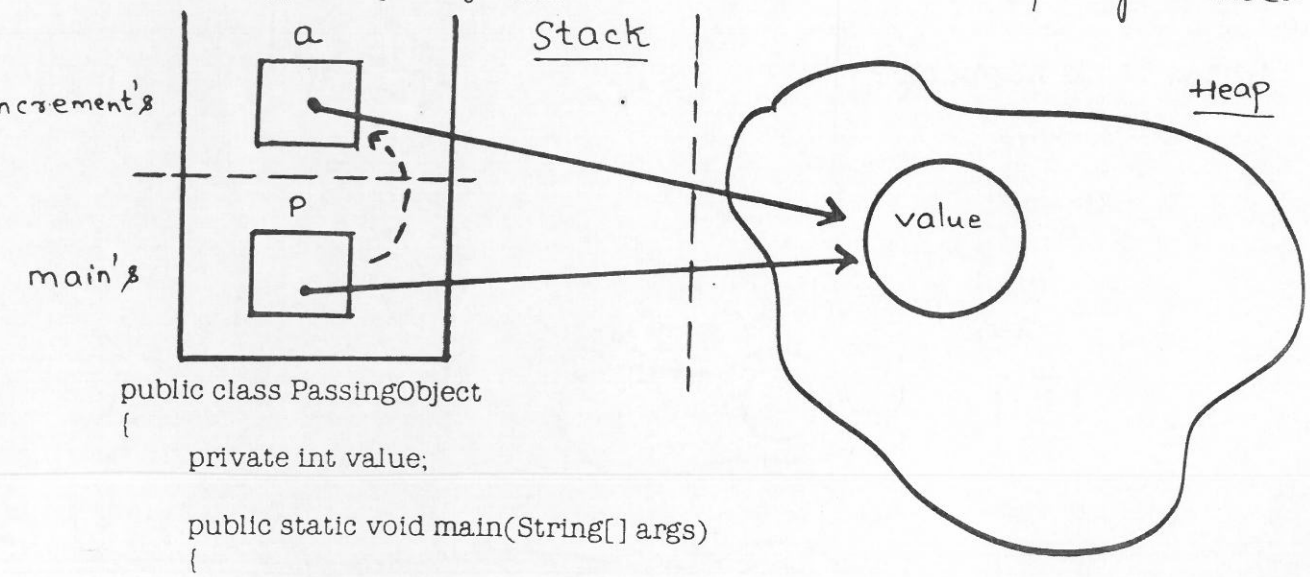


c. Monkey m1 = new Monkey( );
Monkey m2 = new Monkey( );
m2 = m1;



Passing objects to methods :

- Java supports "pass-by-value"




```
public class PassingObject
{
    private int value;

    public static void main(String[] args)
    {
        PassingObject p = new PassingObject( );
```

```
        p.value = 5;
        System.out.println("Before calling: " + p.value);
        increment(p);
        System.out.println("After calling: " + p.value);
    }
    public static void increment( PassingObject a)
    {
        a.value++;
    }
}
```

Output:

Before calling: 5
After calling: 6

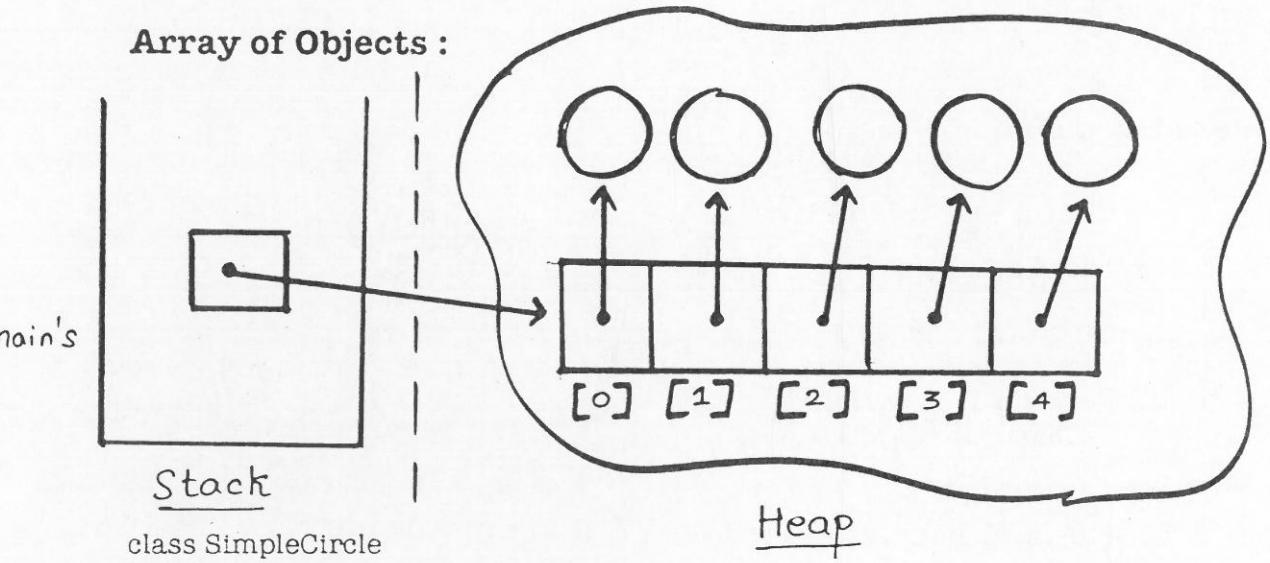
Guess the output:

1.

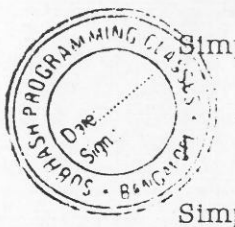
```
public class CheckPoint
{
    public static void main( String [] args )
    {
        Demo d1 = new Demo( 10 );
        Demo d2 = new Demo( );

        copy1(d1, d2);
        System.out.println( "After copy1: d1 = " + d1.d + " d2 = " + d2.d );

        copy2(d1, d2);
        System.out.println("After copy2: d1 = " + d1.d + "d2 = " + d2.d );
    }

    public static void copy1( Demo x, Demo y )
    {
        Demo temp = x;
        x = y;
        y = temp;
    }

    public static void copy2( Demo x, Demo y )
    {
        int temp = x.d;
        x.d = y.d;
        y.d = temp;
    }
}
```

```
class Demo
{
    int d;

    Demo( int newD )
    {
        d = newD;
    }
    Demo( )
    {
    }
}
```

Array of Objects :




```
class SimpleCircle
{
    double radius;

    SimpleCircle( )
    {
        radius = 1;
    }

    SimpleCircle( double newRadius )
    {
        radius = newRadius;
    }

    double getArea( )
    {
        return Math.PI * radius * radius;
    }

    double getPerimeter( )
    {
```



```
        return 2 * Math.PI * radius;
    }
}

public class SimpleCircleTest
{
    public static void main( String [] args )
    {
        SimpleCircle [ ] c  = new SimpleCircle [5];

        for( int i = 0; i < 5; i++ )
            c[i] = new SimpleCircle(i);

        for( int i = 0; i < 5; i++ )
        {
            System.out.println( "Area of " + i + "th Circle is " + c[i].getArea( ) );
            System.out.println( "Perimeter of " + i + "th Circle is " + c[i].getPerimeter( ) );
            System.out.println();
        }
    }
}
```

Output:

Area of 0th Circle is 0.0
Perimeter of 0th Circle is 0.0

Area of 1th Circle is 3.141592653589793
Perimeter of 1th Circle is 6.283185307179586

Area of 2th Circle is 12.566370614359172
Perimeter of 2th Circle is 12.566370614359172

Area of 3th Circle is 28.274333882308138
Perimeter of 3th Circle is 18.84955592153876

Area of 4th Circle is 50.26548245743669
Perimeter of 4th Circle is 25.132741228718345

What is 'this' ?

Fill in the blanks:

```
class PrintMyData
{
    int a;
    int b;
```

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

```
    PrintMyData( )
    {
        a = 6;
        b = 7;
    }

    void Display( )
    {

        System.out.println( "Display a = " + _____ );
        System.out.println( "Display b = " + _____ );

    }
}

public class PrintMyDataTest
{
    public static void main( String [] args )
    {
        PrintMyData p = new PrintMyData( );

        p.Display( );

        System.out.println( "Main a = " + _____ );
        System.out.println( "Main b = " + _____ );
    }
}
```

Output:

```
Display a = 6
Display b = 7
Main a = 6
Main b = 7
```

How to call a parameterised constructor from a no-arg constructor?

- **'this( )'** should be the first statement in a constructor

```
class Monkey
{
    int a;
    int b;

    Monkey( )
    {
        this( 5, 6 );
```

```
    }
    Monkey( int x, int y )
    {
        a = x;
        b = y;
        System.out.println( "I am inside the parameterized constructor" );
    }
    void Display( )
    {
        System.out.println( "a = " + a + " b = " + b );
    }
}

public class ConsFromCons
{
    public static void main( String [] args )
    {
        Monkey m = new Monkey();
        m.Display( );
    }
}
```

Guess the output:

```
public class Test
{
    private int i = 0;
    private int j = 0;

    public static void main( String [] args )
    {
        int i = 2;
        int k = 3;

        System.out.println( "i is" + i );
        System.out.println( "j is" + j );
        System.out.println( "k is " + k );
    }
}
```

2.

```
class Monkey
{
    int a;
    int b;
```

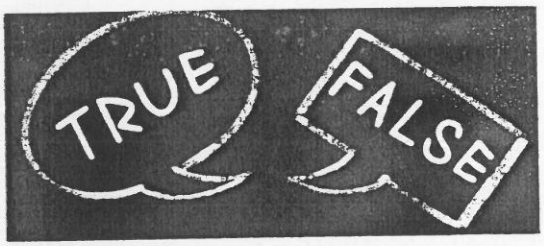
```
        Monkey( )
        {
            this( 5, 6 );
        }
        Monkey( int a, int b )
        {
            a = a;
            b = b;
            System.out.println( "I am inside the parameterized constructor" );
        }
        void Display( )
        {
            System.out.println( "a = " + a + " b = " + b );
        }
}

public class ConsFromCons
{
    public static void main( String [] args )
    {
        Monkey m = new Monkey();
        m.Display( );
    }
}
```

3.

```
class Monkey
{
    int a;
    int b;

    Monkey( int a, int k )
    {
        a = this.a;
        b = k;
    }
    void setValues( int a, in b)
    {
        a = this.a;
        b = this.b
    }
}

public class ConsFromCons
{
    public static void main( String [] args )
    {
        Monkey m = new Monkey( 5, 6);
        m.setValues(1, 3);
```

```
    }
}
```

State True or False:



1. 'this' contains the reference of the object with which the method is called :
2. 'this( )' should be the first statement in the constructor :
3. 'private' data can be accessed outside the class :
4. 'public' data cannot be accessed outside the class :
5. constructors can have return type :
6. constructors are called during the object creation :
7. Objects are created on the stack :
8. The programmer has to manually remove the objects from heap once its use is over :
9. Java supports pass by value :
10. Object is a blue-print for a class :

Fill in the blanks:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW___	Y ___

1. Object is something that occupies _____, has a _____ and have a _____ .
2. The 'state' of an object is changed by the '_____'.
3. When an object is getting created on the _____, the _____ gets automatically called.
4. The class that has a _____ method is called as a launcher class.
5. If I do not want my object reference to point to any object, then, I initialise my object reference to _____ .
6. Default value of boolean instance variable is _____ .

Match the following:

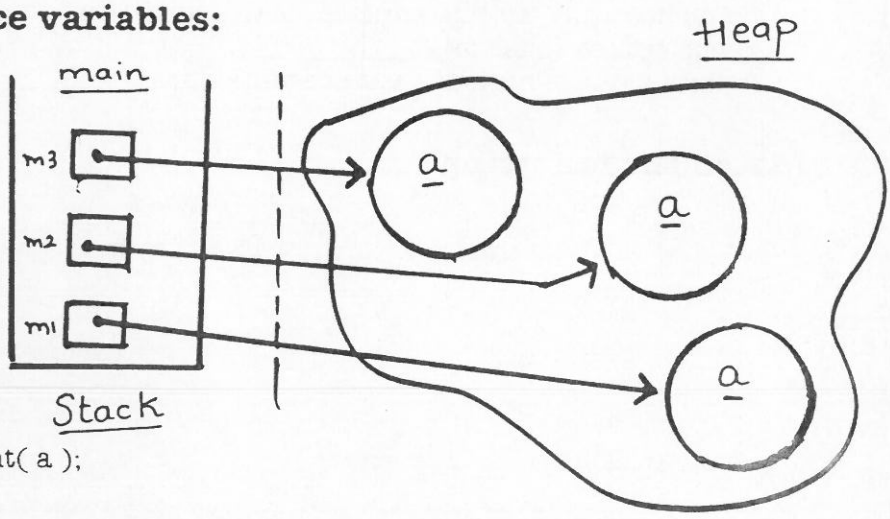
instance variables	should be the first statement in a constructor
this( )	are present on the heap
garbage collector	does not have a return type
class	0.0
default value of float instance variable	a blue print for an object
constructors	automatically deallocates unused memory

All about 'static' in Java

- Static instance variables are part of the class, not part of an object
- Static members are accessed through class names.
- Non-static variables are 1 per instance. Static variables are 1 per class.
- Non-static members cannot be accessed through static methods.
- Non-static methods cannot be called from static methods.
- All static variables are guaranteed to be initialised before any other object of that class is created and before any static method of the class runs.

Non-Static instance variables:

```
class Monkey
{
     int a;

    void increment( )
    {
         a++;
    }

    void display( )
    {
         System.out.print( a );
    }
}
```

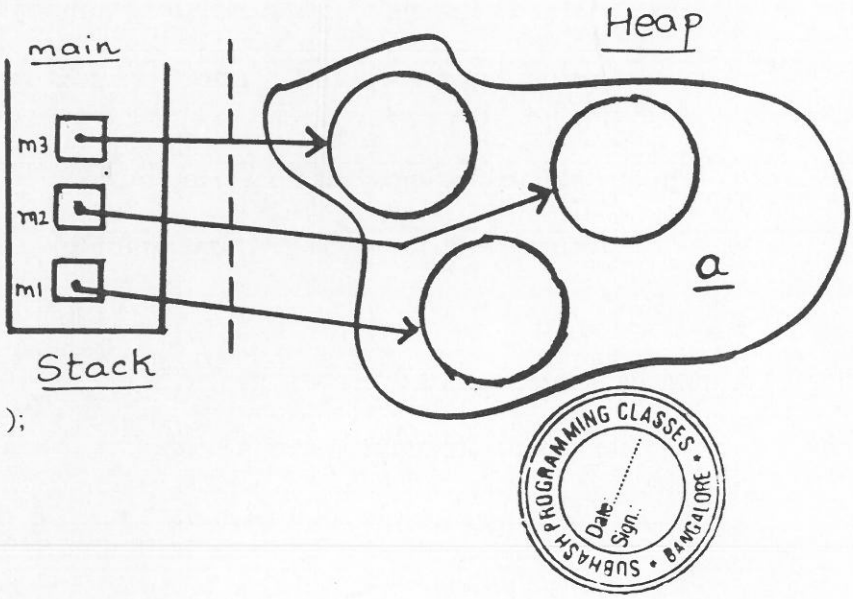


```
public class StaticTest
{
    public static void main( String [] args )
    {
        Monkey m1 = new Monkey( );
        m1.increment( );
        m1.display( );

        Monkey m2 = new Monkey( );
        m2.increment( );
        m2.display( );

        Monkey m3 = new Monkey( );
        m3.increment( );
        m3.display( );

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

Output:

'static' instance variables:

```
class Monkey
{
    static int a;

    void increment( )
    {
        a++;
    }

    void display( )
    {
        System.out.print( a );
    }
}

public class StaticTest
{
    public static void main( String [] args )
    {
        Monkey m1 = new Monkey( );
```



```
        m1.increment( );
        m1.display( );

        Monkey m2 = new Monkey( );
        m2.increment( );
        m2.display( );

        Monkey m3 = new Monkey( );
        m3.increment( );
        m3.display( );

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

Output:

'static' methods has nothing to do with instance variables !

```
class SubhashMath
{
    public static long max( long num1, long num2 )
    {
        return (num1 > num2) ? num1 : num2;
    }

    public static long min( int num1, int num2 )
    {
        return (num1 < num2) ? num1 : num2;
    }
}

public class SubhashMathTest
{
    public static void main( String [] args )
    {
        System.out.println(SubhashMath.max( 5, 6 ));
        System.out.println(SubhashMath.min( 5, 6));
    }
}
```

- All methods in the standards '**Math**' class are **static** methods. You cannot create an instance of a **Math** Class. '**Math**' class do not have any instance variables.

How can I declare a global variable as we do in "C" ?

public **static final** double PI = 3.141592653589793;

An example for a global variable usage using pubic, static, final:

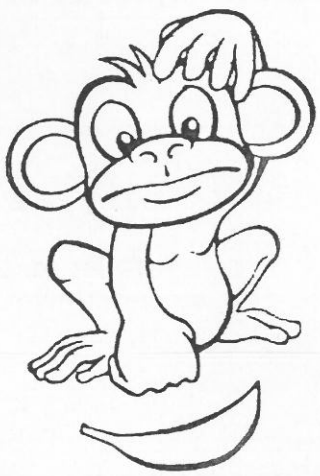
```
class GlobalData
{
    public static final int VAL;

    static
    {
        VAL = (int)(Math.random( ) * 10);
    }
}

public class GlobalDataTest
{
    public static void main( String [ ] args )
    {
        int Result = GlobalData.VAL * 4 * 5;
        System.out.println( Result );
    }
}
```

Static initializer block

Guess the output:

1.

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



2.

```
class StaticTest
{
    public static void fun( )
    {
        System.out.println( "Hello" );
    }
}
class Test
{
    public static void main( String [] args )
    {
        StaticTest s = new StaticTest( );
        s.fun( );
    }
}
```


Do you think we need to make any changes in the below code ?

```
public class Test
{
    public int factorial( int n )
    {
        int result = 1;
        for( int i = 1; i <= n; i++ )
            result *= i;
        return result;
    }
}
```

Few examples of Standard Java Library Classes:

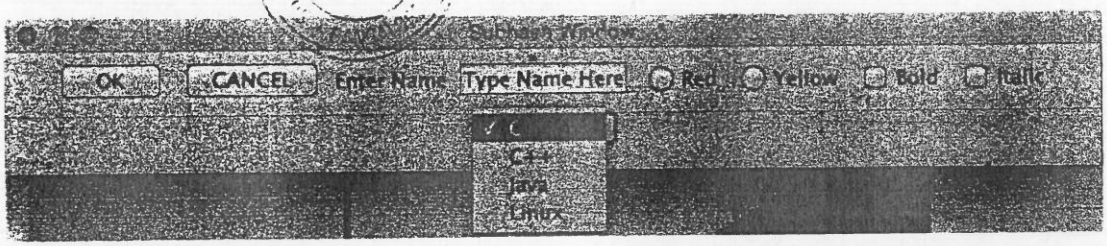
```
import javax.swing.*;

public class JavaClass
{
    public static void main( String [] args )
    {
        java.util.Date date = new java.util.Date( );
        System.out.println( date.getTime() );
        System.out.println( date.toString() );

        java.util.Random r = new java.util.Random( );
        System.out.println( r.nextInt( ));
        System.out.println( r.nextInt(1000));
```

```
        System.out.println( r.nextDouble( ));
        System.out.println( r.nextFloat( ));
        System.out.println( r.nextBoolean( ));

        JButton jb1 = new JButton( "OK" );
        JButton jb2 = new JButton( "CANCEL" );

        JLabel jl = new JLabel( "Enter Name" );

        JTextField ji = new JTextField("Type Name Here");

        JRadioButton jr1 = new JRadioButton("Red");
        JRadioButton jr2 = new JRadioButton("Yellow");

        JCheckBox jc1 = new JCheckBox("Bold");
        JCheckBox jc2 = new JCheckBox("Italic");

        JComboBox jcb = new JComboBox( new String [ ] { "C", "C++", "Java",
"Linux" } );

        JPanel jp = new JPanel();
        jp.add(jb1);
        jp.add(jb2);
        jp.add(jl);
        jp.add(ji);
        jp.add(jr1);
        jp.add(jr2);
        jp.add(jc1);
        jp.add(jc2);
        jp.add(jcb);

        JFrame frame = new JFrame( );
        frame.add(jp);
        frame.setTitle("Subhash Window" );
        frame.setSize(700,100);
        frame.setLocation(200,100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

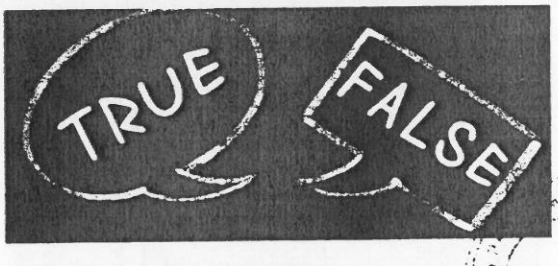
Output:

```
$ java JavaClass
1466085427953
Thu Jun 16 19:27:07 IST 2016
-515814574
6
0.798715197541864
```

0.01757741
false
$



State True or False :



1. non-static methods cannot be called from static methods :
2. 'Math' class does not contain instance variables :
3. All static variables are guaranteed to be initialised before any other object of that class is created and before any static method of the class runs :
4. **'static'** methods can be called via class name or through an object reference :
5. **static** instance variables are shared by all the objects of a class :

Fill in the blanks :

1. Global scope to a variable can be achieved through _____.
2. The **static final** variable can be initialised through _____ block.
3. **static** methods called using _____ .
4. non-static objects are part of individual objects and **static** variables are part of a _____ .

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW___	Y ___

The 'String' of Java

- A String object is immutable, its contents cannot be changed.

How to create a string ?

a. String s1 = new String("Subhash Programming Classes");

b. String s1 = "Subhash Programming Classes" ;

c. char [ ] arr = { 'S', 'u', 'b', 'h', 'a' , 's', 'h' };
String s1 = new String (arr);

d. String s1 = new String(new char [] { 'S', 'u', 'b', 'h', 'a', 's', 'h' });

How to concatenate strings ?

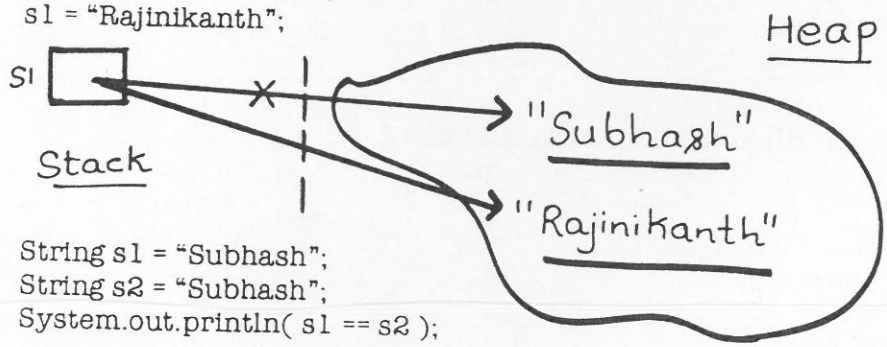
a. String s1 = "Subhash" + " Programming" + " Classes";

b. String s1 = "Subhash" + " Programming";
String s2 = " Classes";
String s3 = s1 + s2;

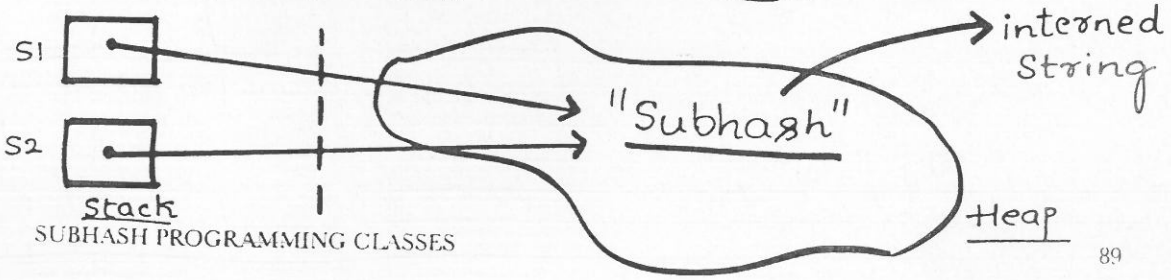
c. String s1 = "Subhash" + " Programming";
String s2 = " Classes";
s1 = s1.concat(s2);

What do you mean by 'interned' strings?

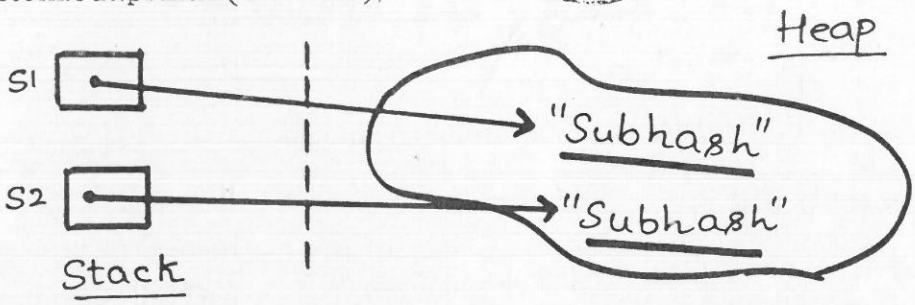
String s1 = "Subhash";
s1 = "Rajinikanth";



String s1 = "Subhash";
String s2 = "Subhash";
System.out.println(s1 == s2);



```
String s1 = new String ("Subhash");
String s2 = "Subhash";
System.out.println( s1 == s2 );
```



How to check the equality of individual characters in two string ?.

- == operator checks only if 's1' and 's2' refer to the same object. It does not check the contents of the string object.
- To compare the contents, use '**equals**' method.

```
String s1 = "Subhash";
String s2 = new String( "Subhash" );
String s3 = "Sukhash";
System.out.println( s1.equals(s2));
System.out.println( s1.equals(s3));
```

How to compare two strings ?

```
String s1 = new String( "subhash" );
String s2 = new String( "Subhash" );
System.out.println( s1.compareTo(s2));
```

How to find the length of a string ?

```
String s1 = "Subhash";
System.out.println(s1.length());
```

How to fetch a single character from a string ?

```
String s1 = "Subhash";
System.out.println(s1.charAt(0));
System.out.println(s1.charAt(6));
```

How to replace characters in a string ?

```
String s1 = "Subhash";
s1 = s1.replace('a', 'k');
System.out.println(s1);
s1 = s1.replaceFirst("h", "b");
System.out.println(s1);
s1 = s1.replaceAll("Sub", "aaa");
System.out.println(s1)
```


```
String s1 = "S+u$b#hash";
s1 = s1.replaceAll( "[+$#]", "NNN");
System.out.println(s1);
```

How to find substring in a string ?

```
String s1 = "Subhash Programming Classes";
System.out.println( s1.substring(8, 11));
```

How to find the index of a character or substring in a string ?

```
String s1 = "Subhash Programming Classes";
System.out.println( s1.indexOf('s' ));
System.out.println( s1.indexOf('s', 6));
System.out.println( s1.lastIndexOf('s'));
System.out.println( s1.lastIndexOf("gram"));
```

How to split a string ?


```
String s1 = "Subhash# Loves# Programming";

String [] words = s1.split("#");
for( int i = 0; i < words.length; i++ )
{
    System.out.print( words[i] );
}
System.out.println( );


String s1 = "Subhash# Loves? Programming: With, Java";

String [] words = s1.split("[#?:,]");
for( int i = 0; i < words.length; i++ )
{
    System.out.print( words[i] );
}
System.out.println( );
```

How to delete the preceding and trailing white spaces in a string ?

```
String s1 = "        Subhash Loves Programming With Java";

System.out.println(s1);
s1 = s1.trim( );
System.out.println(s1);
```

How to do pattern matching in a string ?

```
String s1 = "Java is my favourite";
String s2 = "2345/23/4/123";
System.out.println( s1.matches( "Java.*"));
System.out.println( s1.matches( "java.*"));
System.out.println( s2.matches( "\\d{4}/\\d{2}/\\d/\\d{3}" ));
```

How to convert the case of a string ?

```
String s1 = "Subhash";
s1 = s1.toUpperCase();
System.out.println(s1);
s1 = s1.toLowerCase();
System.out.println(s1);
```

'StringBuilder' class:

What is the difference between 'String' and 'StringBuilder' class ?

- The **StringBuilder** and **String** classes are similar to the String class except that the String class is immutable.

How to append to a string ?

```
StringBuilder s = new StringBuilder( "Subhash" );
s.append( " Programming" );
s.append( " Classes" );
System.out.println( s );
```

How to delete characters from a string ?

```
StringBuilder s = new StringBuilder( "Subhash" );
s.delete( 3, 6 );
System.out.println(s);
s.deleteCharAt(1);
System.out.println(s);
```

How to replace in a string ?

```
StringBuilder s = new StringBuilder( "Subhash" );
s.replace(2,5,"me");
System.out.println( s );
```

How to set a character in a string ?

```
StringBuilder s = new StringBuilder( "Subhash" );
s.setCharAt(0,'s');
System.out.println( s );
```

How to insert characters in a string ?

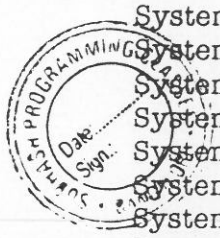
```
StringBuilder s = new StringBuilder( "Sash" );
s.insert(1,"ubh");
System.out.println( s );
```

How to reverse a string ?

```
StringBuilder s = new StringBuilder( "Subhash" );
s.reverse( );
System.out.println( s );
```

Guess the output:

```
1.  public class Test
    {
        public static void main( String [] args )
        {
            String s1 = "Welcome to Java";
            String s2 = s1;
            String s3 = new String( "Welcome to Java" );
            String s4 = "Welcome to Java";

            System.out.println( s1 == s2 );
            System.out.println( s2 == s3 );
            System.out.println( s1.equals(s2));
            System.out.println( s2.equals(s3));
            System.out.println( s1.compareTo(s2));
            System.out.println( s2.compareTo(s3));
            System.out.println( s1 == s4 );
            System.out.println( s1.charAt(0));
            System.out.println( s1.indexOf('j'));
            System.out.println( s1.indexOf("to"));
            System.out.println( s1.lastIndexOf('a'));
            System.out.println( s1.lastIndexOf("o", 15));
```



```
        System.out.println( s1.length( ));
        System.out.println( s1.substring(5));
        System.out.println( s1.substring(5, 11));
        System.out.println( s1.toLowerCase());
        System.out.println( s1.toUpperCase());
        System.out.println( "  Welcome  ".trim());
        System.out.println( s1.replace('o', 'T'));
        System.out.println( s1.replaceAll("o", "T"));
        System.out.println( s1.replaceFirst("o", "T"));
    }
}
```

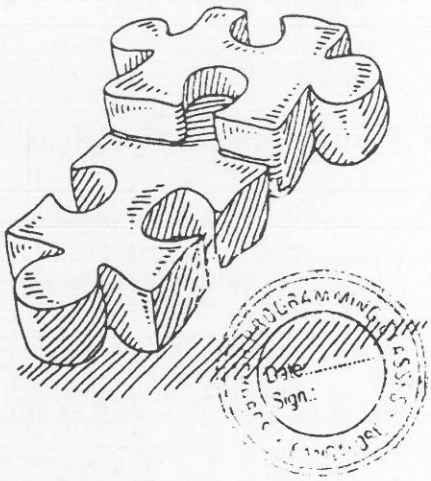
2.


```
class Text
{
    String text;

    Text( String s )
    {
        String text = s;
    }
}
public class Test
{
    public static void main( String [] args )
    {
        Text test = new Text( "Hello" );
        System.out.println( test.text.toUpperCase( ));
    }
}
```

3.

```
public class Test
{
    public static void main( String [] args )
    {
        System.out.println( "Hi, XYZ, good".matches("ABC"));
        System.out.println( "Hi, RSS, good".matches("RSS.*"));
        System.out.println( "Hi, MSS, good".matches(".*MSS.*"));
        System.out.println( "S,K;U".replaceAll(";","&"));

        String [] tokens = "S, K;U".split( "[;]");
        for( int i = 0; i < tokens.length; i++ )
        {
            System.out.println( tokens[i] + " " );
        }
    }
}
```

Do it yourself - Right Now !



1. Let s1 be " Subhash " and s2 be " subhash ". Write the code for the following statements.

- Check whether s1 is equal to s2 and assign the result to a Boolean variable isEqual.

- Check whether s1 is equal to s2, ignoring case, and assign the result to a Boolean variable isEqual.

- Compare s1 with s2 and assign the result to an int variable x.

- Compare s1 with s2, ignoring case, and assign the result to an int variable x.

- Check whether s1 has the prefix AAA and assign the result to a Boolean variable b.

- Check whether s1 has the suffix AAA and assign the result to a Boolean variable b.

- Assign the length of s1 to an int variable x.

- Assign the first character of s1 to a char variable x.

- Create a new string s3 that combines s1 with s2.

- Create a substring of s1 starting from index 1.

- Create a substring of s1 starting from index 1 to index 4.

- Create a new string s3 that converts s1 to uppercase.

- Create a new string s3 that converts s1 to lowercase.

- Create a new string s3 that trims blank spaces on both ends of s1.

- Replace all occurrences of the character 'e' with 'E' in s1 and assign the new string to s3.

- Split "Welcome to Java and HTML" into an array tokens delimited by a space.

- Assign the index of the first occurrence of the character 'e' in s1 to an int variable x.

- Assign the index of the last occurrence of the string "abc" in s1 to an int variable x.

- WAP to find whether a given string is palindrome or not.

Command Line Arguments:

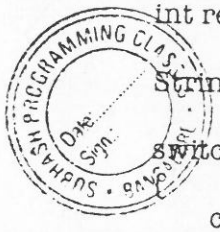
```
public class CLA
{
    public static void main( String [] args )
    {
        for( int i = 0; i < args.length; i++ )
        {
            System.out.println( args[i] );
        }
    }
```

```
}
```

```
$ javac CLA.java
$ java CLA
(no output)
$java CLA subhash programming classes
subhash
programming
classes
```

Simple Calculator using CLA:

```
public class SimpleCalculator
{
    public static void main( String [] args )
    {
        if( args.length != 1 )
        {
            System.out.println( "USAGE:
java SimpleCalculator \" operand1 operator
operand2\"");
            System.exit(0);
        }

        int result = 0;

        String [] tokens = args[0].split(" ");

        switch( tokens[1].charAt(0))

            case '+' : result = Integer.parseInt(tokens[0]) +
Integer.parseInt(tokens[2]);
                    break;

            case '-' : result = Integer.parseInt(tokens[0]) -
Integer.parseInt(tokens[2]);
                    break;

            case '*' : result = Integer.parseInt(tokens[0]) *
Integer.parseInt(tokens[2]);
                    break;

            case '/' : result = Integer.parseInt(tokens[0]) /
Integer.parseInt(tokens[2]);
        }

        System.out.println( tokens[0] + ' ' + tokens[1] + ' ' + tokens[2] + " = " +
result );
```



```
    }
}
```


```
$ java SimpleCalculator
USAGE: java SimpleCalculator " operand1 operator operand2"
$ java SimpleCalculator "3 + 5"
3 + 5 = 8
$ java SimpleCalculator "2 * 5"
2 * 5 = 10
java SimpleCalculator "2 - 6"
2 - 6 = -4
$ java SimpleCalculator "2 / 6"
2 / 6 = 0
$
```

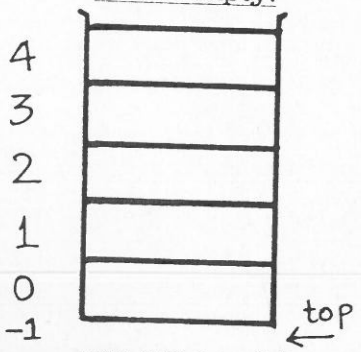
Developing a 'stack' class

- Get the requirements right. (Although this is not possible in the practical world :))
- Always start with a pictorial representation.
- Start building the code piece by piece.

Step 1: Identify the object state
Step 2: Identify the object behaviour
Step 3: Write the Driver Class / Launcher Class / Application Class

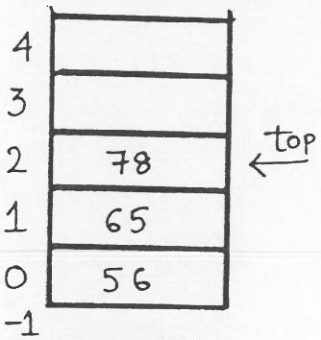
Pictorial Understanding:

Stack Empty:



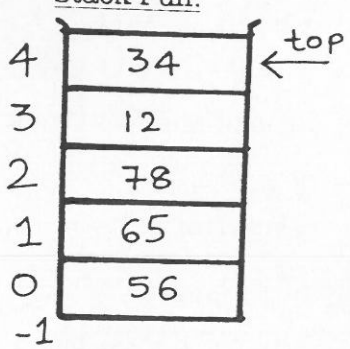
pop not possible
push possible
Display not possible

Stack Half-Filled:



pop possible
push possible
Display possible

Stack Full:



pop possible
push not possible
Display possible

Object State:

- top
- array
- CAPACITY

Object Behaviour:

- push( )
- pop( )
- display( )
- IsStackFull( )
- IsStackEmpty( )

Implementation:

```
class Stack
{
    private int top;
    private int [] array;

    public static final int CAPACITY = 5;

    public Stack( )
    {
        top = -1;
        array = new int [CAPACITY];
    }

    public boolean IsStackFull( )
    {
        return top == (array.length - 1 );
    }

    public boolean IsStackEmpty( )
    {
        return top == -1;
    }

    public void push( int num )
    {
        ++top;
        array[top] = num;
    }
    public int pop( )
    {
        int num = array[top];
        --top;
```

```
        return num;
    }
    public void display( )
    {
        for( int i = 0; i <= top; i++ )
        {
            System.out.print( array[i] + " " );
        }
    }
}

public class StackTest
{
    public static void main( String [] args )
    {
        Stack s = new Stack( );
        Scanner in = new Scanner( System.in );

        while(true)
        {
            if( s.IsStackEmpty())

            {
                System.out.println( "Stack is Empty !!!" );
                System.out.println( "Enter a <num> to push or \'e\' to EXIT" );
            }
            else if( s.IsStackFull())
            {
                System.out.println( "Stack is Full !!!" );
                System.out.println( "Enter \'p\' to POP or \'e\' to EXIT or \'d\'
to DISPLAY " );
            }
            else
            {
                System.out.println( " Enter a <num> to push or \'p\' to POP or
\'e\' to EXIT or \'d\' to DISPLAY" );
            }

            String option = in.next( );
            if( option.equals("p"))
            {
                System.out.println( s.pop());
            }
            else if( option.equals( "e" ))
            {
                System.exit(0);
            }
            else if( option.equals("d"))
            {
```

```
                    s.display();
                    System.out.println( );
                }
                else
                {
                    int number = Integer.parseInt(option);
                    s.push(number);
                }
            }
        }
    }
```

Output:

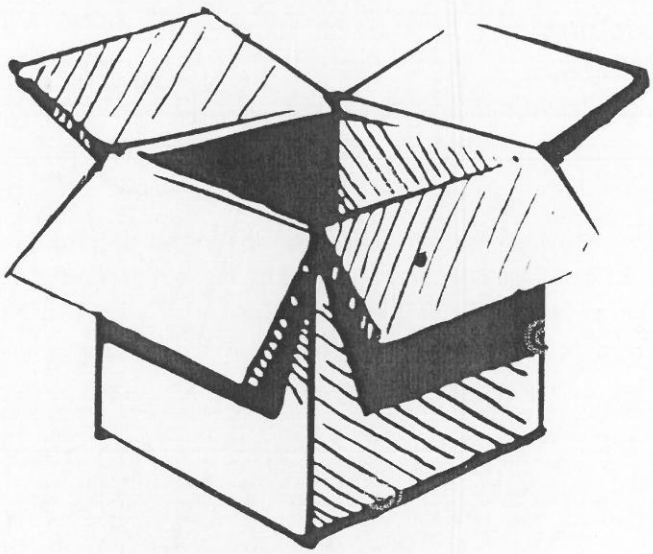
```
$ java StackTest
Stack is Empty !!!
Enter a <num> to push or 'e' to EXIT
56
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
65
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
78
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
12
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
34
Stack is Full !!!
Enter 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
p
34
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
p
12
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
d
56 65 78
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
23
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
d
56 65 78 23
 Enter a <num> to push or 'p' to POP or 'e' to EXIT or 'd' to DISPLAY
e
$
```

Auto-boxing & Auto-Unboxing

Wrapper Classes:

Boolean
Character
Byte Short Integer Long
Float Double

- Wrapping a primitive type into an object type is known **Boxing.** The reverse is **unboxing**.




```
Integer i = new Integer(5); // Boxing or Wrapping
int j = i.intValue( );          // Unboxing or Unwrapping

Integer i = 5; //Auto-boxing
int j = i;      // Auto-unboxing
```

How to convert a numeric integer string into a integer variable ?

```
int i = Integer.parseInt("5");
```

How to convert an integer value into a numeric string ?

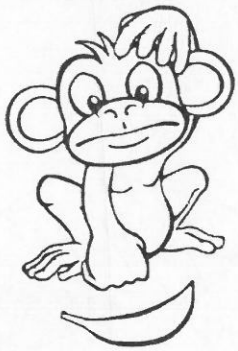
```
int i = 5;
String num = Integer.toString( 5 );
```

How to convert a numeric boolean string into a boolean variable ?

```
boolean b = new Boolean("true").booleanValue( );
```

How to convert an integer object into a numeric string ?

```
Integer i = new Integer(5);
String num = i.toString( );
```



Guess the Output:

1. String t = "two";
2. int y = Integer.parseInt(t);
   System.out.println(y);

Inheritance & Polymorphism

Design level discussion on Inheritance:

Let me introduce two professors:

a. Prof. Kamakshi

Expert in C. No idea about C++/Java

b. Prof.Subbulakshmi

Expert in C, C++ and Java

1. Designing a **Teacher** Class

Step-1: Initial Design.

Prof. Kamakshi's Style - A Procedure Oriented Approach :

```
doTeaching( subjectType )
{
        //teach the specific subject in the classroom
}
giveAssignment( subjectType )
{
        //use subjectType to see which subject to give assignments
}
```

Prof. Subbulakshmi's Style - An Object Oriented Approach :

```
class MathsTeacher
{
        doTeaching( )
        {
        //teach mathematics in classroom
        }

        giveAssignment( )
        {
        //Give mathematics assignment
        }
}
```

```
class PhysicsTeacher
{
        doTeaching( )
        {
      //teach physics in classroom
        }

        giveAssignment( )
        {
        //give physics assignment
        }
}
```

```
class ChemistryTeacher
{
        doTeaching( )
        {
        //teach chemistry in classroom
        }

        giveAssignment( )
        {
        //Give chemistry assignment
        }
}
```

Step-2: When the specification changes. Add **MotivationalTeacher**.

Prof. Kamakshi's Style - A Procedure Oriented Approach :

```
doTeaching( subjectType )
{
        # if teacher not MotivationalTeacher
        //teach the specific subject in the classroom
        #else
        //teach in school garden
}
giveAssignment( subjectType )
{
        //use subjectType to see which subject to give assignments
```

}

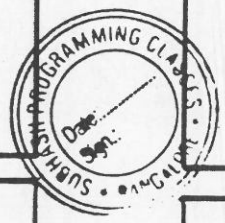
Prof. Subbulakshmi's Style - An Object Oriented Approach :

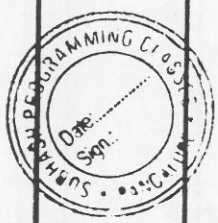
```
class MathsTeacher
{
        doTeaching( )
        {
        //teach mathematics in classroom
        }

        giveAssignment( )
        {
        //Give mathematics assignment
        }
}
```

```
class PhysicsTeacher
{
        doTeaching( )
        {
      //teach physics in classroom
        }

        giveAssignment( )
        {
        //give physics assignment
        }
}
```

```
class ChemistryTeacher
{
        doTeaching( )
        {
        //teach chemistry in classroom
        }

        giveAssignment( )
        {
        //Give chemistry assignment
        }
}
```

```
class MotivationalTeacher
{
        doTeaching( )
        {
        //teach in school garden
        }

        giveAssignment( )
        {
        //give assignment
        }
}
```





Step-3: When the specification changes again. **Teaching location** change.

Prof. Kamakshi's 'troubled' Style - A Procedure Oriented Approach :

```
doTeaching( subjectType )
{
        # if teacher not MotivationalTeacher
        //teach the specific subject in the classroom
        #else if weekday >= Monday <= Wednesday
        //teach in school garden
        #else
        //teach in school seminar hall
}
giveAssignment( subjectType )
```

```
{
        //use subjectType to see which subject to give assignments
}
```

Prof. Subbulakshmi's 'Working' Style - An Object Oriented Approach :

```
class MathsTeacher
{
        doTeaching( )
        {
        //teach mathematics in classroom
        }

        giveAssignment( )
        {
        //Give mathematics assignment
        }
}
```

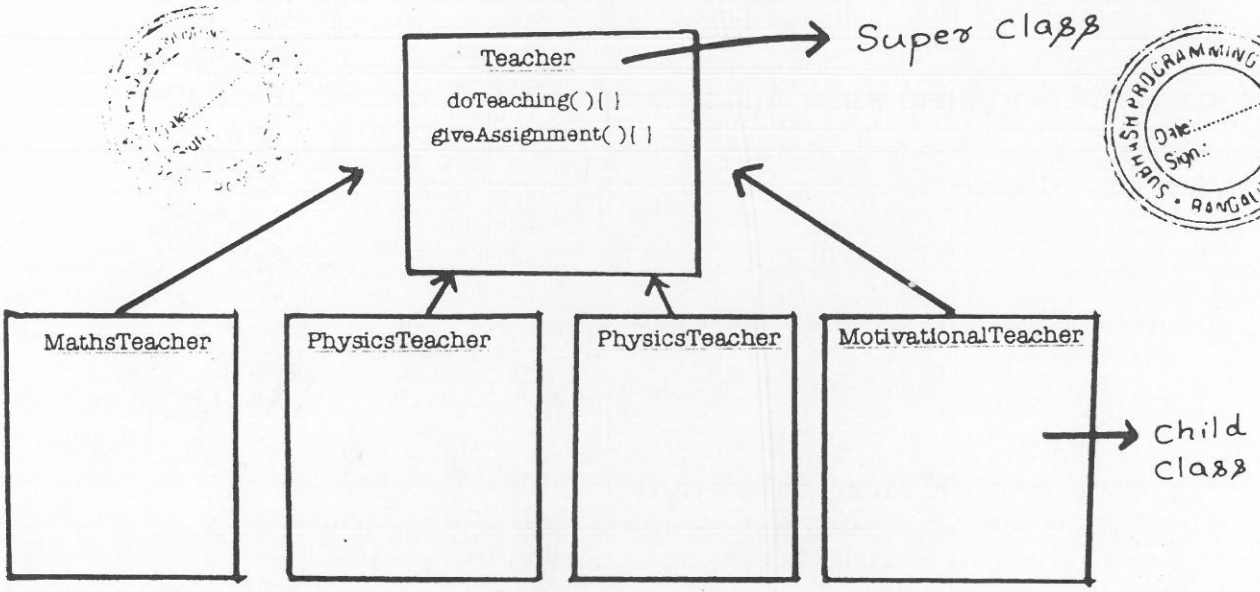
```
class PhysicsTeacher
{
        doTeaching( )
        {
       //teach physics in classroom
        }

        giveAssignment( )
        {
        //give physics assignment
        }
}
```

```
class ChemistryTeacher
{
        doTeaching( )
        {
        //teach chemistry in classroom



        }

        giveAssignment( )
        {
        //Give chemistry assignment
        }
}
```

```
class MotivationalTeacher
{
        doTeaching( )
        {
        #if (mon, tue, wed)
         //teach in garden
        #else
        //teach in seminar hall
        }

        giveAssignment( )
        {
        //give assignment
        }
```

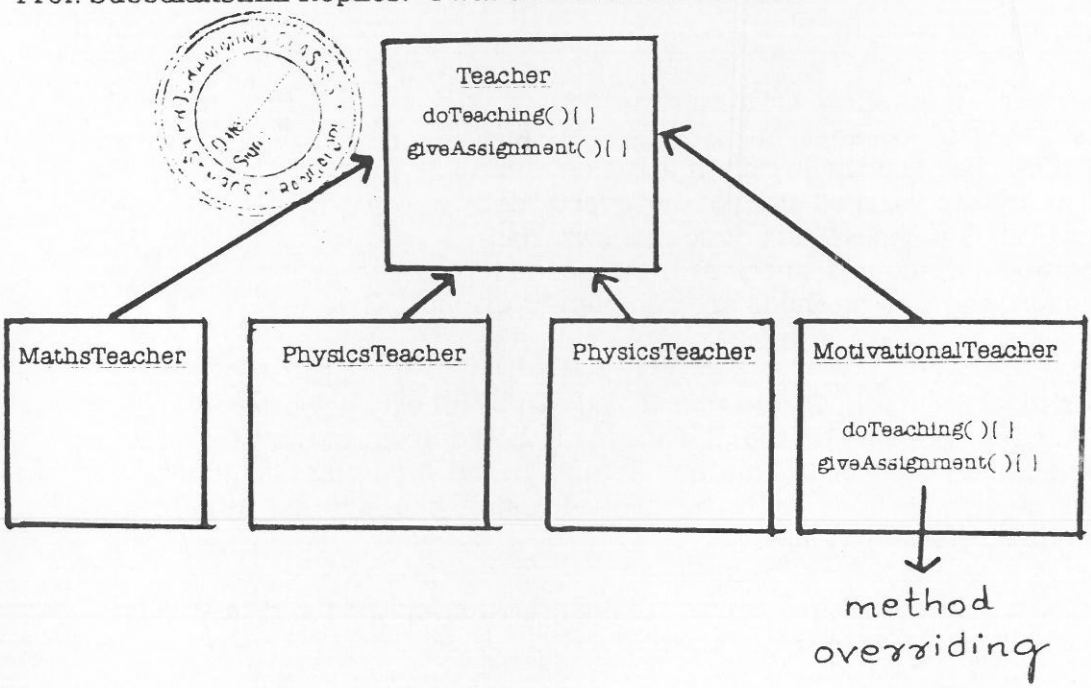
Step-4: Design debate (**A polite fight**) begins. Who wins ?

Prof. Kamakshi Says: "Your code is duplicated"
Prof. Subbulakshmi Replies: No. I have used inheritance.

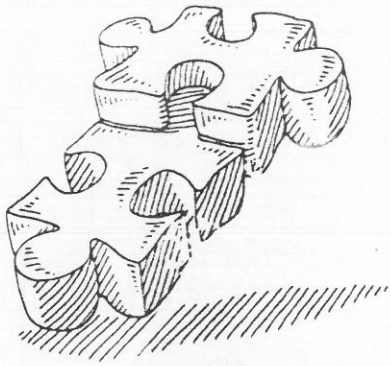


Step-5: Design debate (**A polite fight**) continues. Who wins ?

Prof. Kamakshi Says: "The code for **MotivationalTeacher** is not the same"
Prof. Subbulakshmi Replies: "I will **override** the method"



Do it yourself - Right Now !



Design an Animal Class:

Given Animals: HumanBeing, Lion, Hippopotamus, Tiger, Dog, Cat, Wolf

General Inheritance Structure:

```
class SuperClassName
{

}
class ChildClassName extends SuperClassName
{

}
```

- **'private'** members cannot be inherited
- child class object is created only after base class object is constructed.
- '**super**( )' calls the super class constructor and it should be the first statement in a constructor.
- You can refer to the base class members in a derived class using **super** keyword.
- Redefining the base class method in a derived class is known as '**method overriding**'.

SPECIAL NOTES:

- To perform method overriding the following points needs to be kept in mind. An instance method can be overridden only if it is accessible. Thus a private method cannot be overridden, because it is not accessible outside its own class. If a method defined in a subclass is private in its superclass, the two methods are completely unrelated.

- Like an instance method, a static method can be inherited. However, a static method cannot be overridden. If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden. The hidden static methods can be invoked using the syntax **SuperClassName.staticMethodName**.

- Arguments and the return type of the overridden method and the overriding method should be exactly same.

Guess the Output:

1.

```
class Base
{
    public int a;

    public Base( )
    {
        a = 6;
    }

    public void CallMe( )
    {
        System.out.println( "Base:Called you" );
    }

    public void display( )
    {
        System.out.println( "a = " + a );
    }
}

class Derived1 extends Base
{
    public int b;

    public Derived1( )
    {
        b = 7;
    }

    public void display( )
    {
        System.out.println( "a = " + a + " b = " + b );
    }
}

class Derived2 extends Derived1
{
    public int c;

    public Derived2( )
    {
        c = 8;
    }

    public void display( )
```

```
    {
        System.out.println( "a = " + a + " b = " + b + " c = " + c );
    }
}

public class Inheritance1
{
    public static void main( String [] args )
    {
        Base b     = new Base();
        Derived1 d1 = new Derived1( );
        Derived2 d2 = new Derived2( );

        b.display( );
        d1.display( );
        d2.display( );

        b.CallMe( );
        d1.CallMe( );
        d2.CallMe( );
    }
}
```

2.

```
class Base
{
    public int a;

    public Base( )
    {
        a = 6;
    }

    public void CallMe( )
    {
        System.out.println( "Base:Called you" );
    }

    public void display( )
    {
        System.out.println( "a = " + a );
    }
}

class Derived1 extends Base
{
    public int b;
```

```
    public Derived1( )
    {
        a = 11;
        b = 7;
    }

    public void display( )
    {
        System.out.println( "a = " + a + " b = " + b );
    }
}

class Derived2 extends Derived1
{
    public int c;

    public Derived2( )
    {
        a = 12;
        b = 13;
        c = 8;
    }

    public void display( )
    {
        System.out.println( "a = " + a + " b = " + b + " c = " + c );
    }
}

public class Inheritance2
{
    public static void main( String [] args )
    {
        Base b    = new Base();
        Derived1 d1 = new Derived1( );
        Derived2 d2 = new Derived2( );

        b.display( );
        d1.display( );
        d2.display( );

        b.CallMe( );
        d1.CallMe( );
        d2.CallMe( );
    }
}
```

3.

```
class Base
{
    public int a;

    public Base( int x )
    {
        a = x;
    }

    public void CallMe( )
    {
        System.out.println( "Called you" );
    }

    public void display( )
    {
        System.out.println( "a = " + a );
    }
}

class Derived1 extends Base
{
    public int b;

    public Derived1( int y )
    {
        super(y);
        b = y;
    }

    public void display( )
    {
        System.out.println( "a = " + a + " b = " + b );
    }
}

class Derived2 extends Derived1
{
    public int c;

    public Derived2( int z )
    {
        super(z);
        c = z;
    }

    public void display( )
    {
```

```
        System.out.println( "a = " + a + " b = " + b + " c = " + c );
    }
}

public class Inheritance3
{
    public static void main( String [] args )
    {
        Base b    = new Base(5);
        Derived1 d1 = new Derived1(6);
        Derived2 d2 = new Derived2(7);

        b.display( );
        d1.display( );
        d2.display( );
    }
}
```

4.

```
class Base
{
    public int a;

    public Base( int x )
    {
        a = x;
    }

    public void CallMe( )
    {
        System.out.println( "Base:Called you" );
    }

    public void display( )
    {
        System.out.println( "a = " + a );
    }
}

class Derived1 extends Base
{
    public int a;
    public int b;

    public Derived1( int y )
    {
        super(y);
```

```
        b = y;
    }

    public void display( )
    {
        System.out.println( "Super's a = " + super.a );
        System.out.println( "a = " + a + " b = " + b );
    }
}

class Derived2 extends Derived1
{
    public int b;
    public int c;

    public Derived2( int z )
    {
        super(z);
        c = z;
    }

    public void display( )
    {
        System.out.println( "Super's a = " + super.a );
        System.out.println( "Super's b = " + super.b );
        System.out.println( "a = " + a + " b = " + b + " c = " + c );
    }
}

public class Inheritance4
{
    public static void main( String [] args )
    {
        Base b     = new Base(5);
        Derived1 d1 = new Derived1(6);
        Derived2 d2 = new Derived2(7);

        b.display( );
        d1.display( );
        d2.display( );
    }
}
```

5.

```
class Base
{
    public void Display( )
    {
```

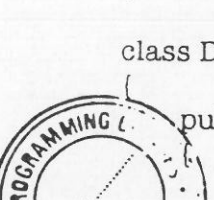
```
            System.out.println( "Base's Display" );
        }
    }

    class Derived extends Base
    {
        public void Display( )
        {
            super.Display( );
            System.out.println( "Derived's Display" );
        }
    }

    public class Sample
    {
        public static void main( String [] args )
        {
            Derived d = new Derived( );
            d.Display( );
        }
    }
```

6.

```
class Base
{
    public void Display( )
    {
        System.out.println( "Base's Display" );
    }
}

class Derived extends Base
{
    public void Display( )
    {
        super.Display( );
        System.out.println( "Derived's Display" );
    }
}

public class Sample
{
    public static void main( String [] args )
    {
        Derived d = new Derived( );
        d.Display( );
    }
}
```

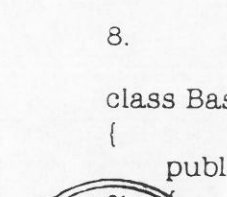
7.

```
class Base
{
    private void Display( )
    {
        System.out.println( "Base's Display" );
    }
}

class Derived extends Base
{
    public void Display( )
    {
        super.Display( );
        System.out.println( "Derived's Display" );
    }
}

public class Sample
{
    public static void main( String [] args )
    {
        Derived d = new Derived( );
        d.Display( );
    }
}
```



8.

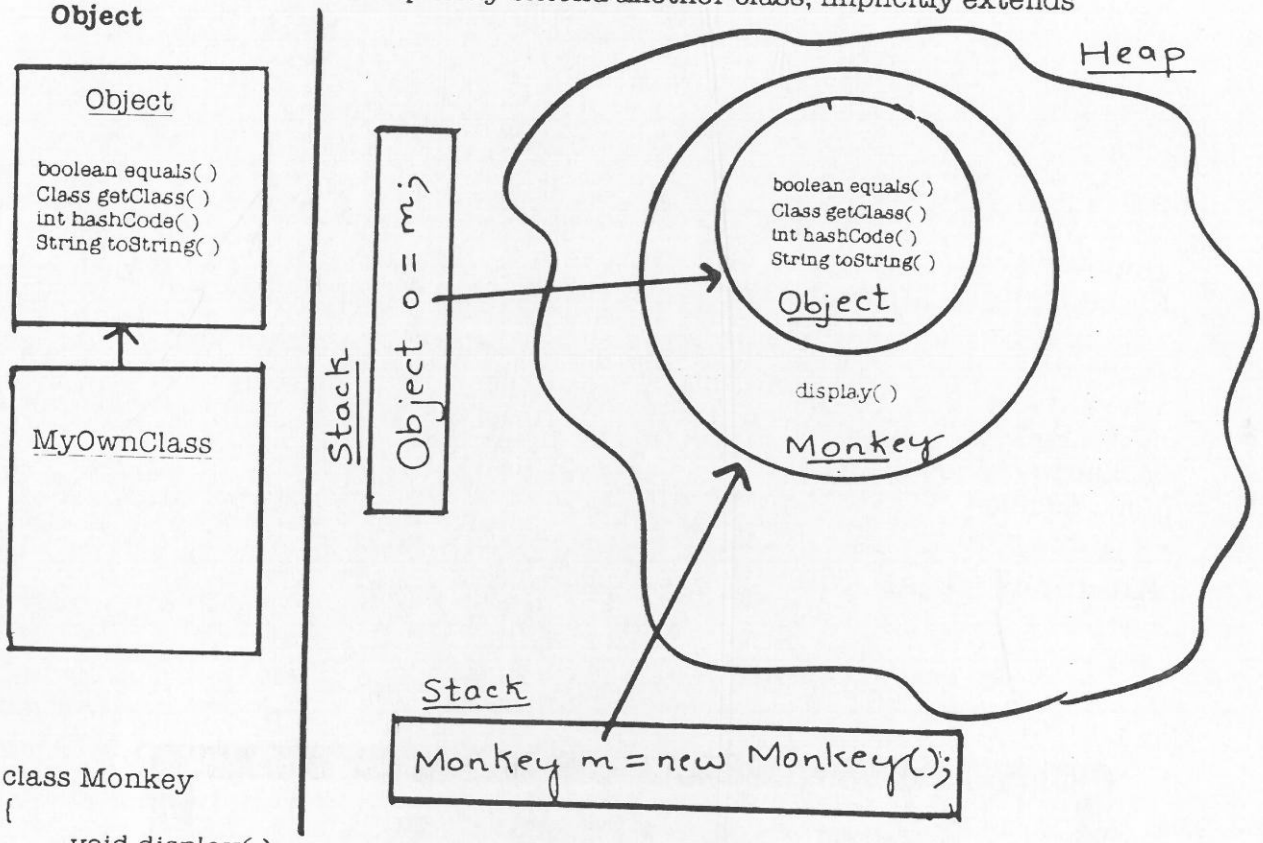
```
class Base
{
    public static void Display( )
    {
        System.out.println( "Base's Display" );
    }
}

class Derived extends Base
{
    public static void Display( )
    {
        super.Display( );
        System.out.println( "Derived's Display" );
    }
}

public class Sample
{
    public static void main( String [] args )
```



```
    {
        Derived d = new Derived( );
        d.Display( );
    }
}
```

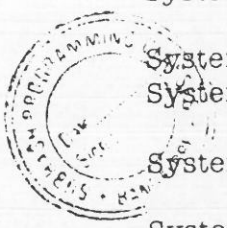
The father of all classes - The 'Object' class:

- Every class in Java extends class **Object** implicitly.
- Any class that doesn't explicitly extend another class, implicitly extends **Object**




```
class Monkey
{
    void display( )
    {
        System.out.println( "Hello Buddies\n");
    }
}

public class Test
{
    public static void main( String [] args )
    {
        Monkey m1 = new Monkey( );
        Monkey m2 = new Monkey( );
        Monkey m3 = m1;
```

```
        System.out.println( m1.hashCode( ));
        System.out.println( m2.hashCode( ));
        System.out.println( m3.hashCode( ));

        System.out.println( m1.equals(m2));
        System.out.println( m1.equals(m3));

        System.out.println( m1.getClass( ));

        System.out.println( m2.toString( ));

        m1.display( );
    }
}
```



```
$ java Test

1829164700
2018699554
1829164700
false
true
class Monkey
Monkey@7852e922
Hello Buddies
```

All about 'final'

1. '**final**' class cannot be extended. The below code results in an error.

```
final class Monkey
{
}
class MonkeyKid extends Monkey
{
}
```

2. '**final**' methods cannot be overridden. The below code results in an error.

```
class Monkey
{
    final void display( )
    {
    }
}

class MonkeyKid extends Monkey
{

    void display( )
    {
    }
}
```

3. '**final**' variable cannot be modified. The below code results in an error.

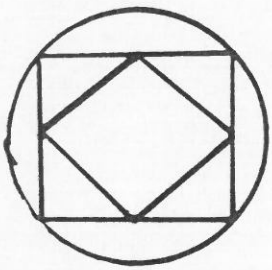
```
public class Test
{
    public static void main( String [] args )
    {
        final int x = 5;
        x = x + 1;
    }
}
```

4. '**final**' instance variable cannot be modified. It should be initialized during its creation. The below code results in an error.

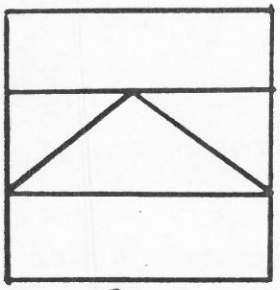
```
public class Monkey
{
    public final int tailLength = 6;

    Monkey( )
    {
        tailLength = 7;
    }
}
```

```
public class Test
{
    public static void main( String [] args )
    {
        final int x = 5;
        x = x + 1;
    }
}
```

Design a code to develop the below figure:



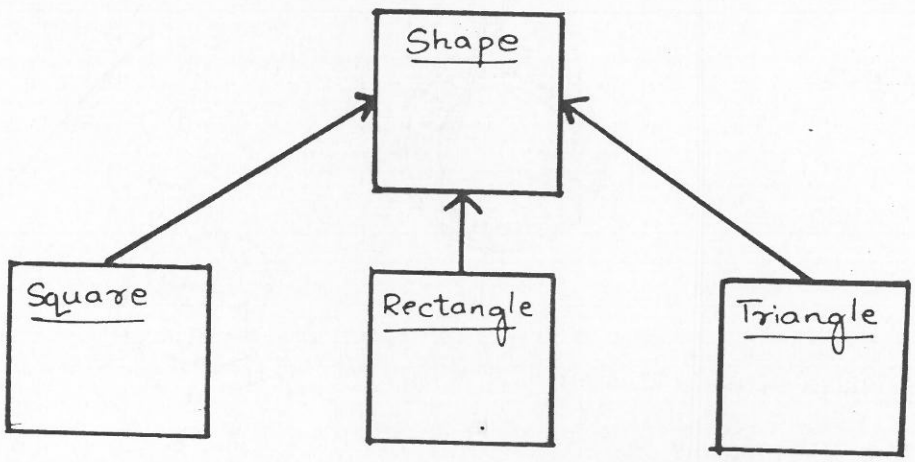
1.



2.

```
class Shape
{
    public void Draw( )
    {
        System.out.println( "Shape Draw" );
    }
}

class Square extends Shape
{
    public void Draw( )
    {
        System.out.println( "Square Draw" );
    }
}

class Rectangle extends Shape
{
    public void Draw( )
    {
        System.out.println( "Rectangle Draw" );
    }
}

class Triangle extends Shape
{
    public void Draw( )
    {
```

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

    public class Inheritance5
    {
        public static void main( String [] args )
        {
            Shape sh = new Shape( );
            sh.Draw( );

            Shape sref = new Square( );
            sref.Draw( );

            Shape rref = new Rectangle( );
            rref.Draw( );

            Shape tref = new Triangle( );
            tref.Draw( );
        }
    }
```



- Shape do not have a particular shape.
- Make the 'Shape' class '**abstract**'.
- You cannot create an instance of an '**abstract**' class.

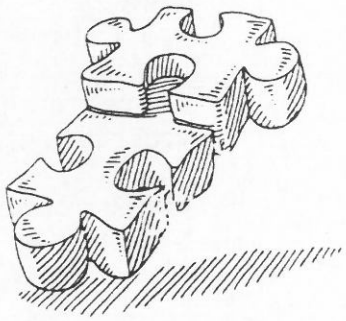
```
abstract class Shape
{
    public void Draw( )
    {
        System.out.println( "Shape Draw" );
    }
}
```

- **Shape sh = new Shape( )** will result in an error in the above code.

```
abstract class Shape
{
    public abstract void Draw( );
}
```

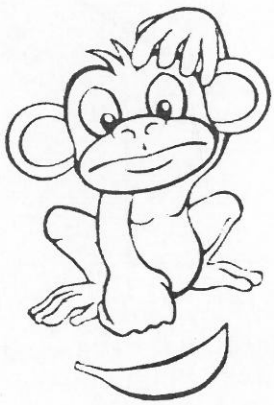
- **'abstract' methods** has to be overridden in the sub-classes, else even the sub-classes are treated as abstract.
- The class that contains **abstract** methods should be marked '**abstract**'.

Do it yourself - Right Now !



- Create a non-abstract class X. (Non-Abstract classes are called **Concrete classes**)
- Add a abstract method f( ) in it.
- Inherit a class Y from X.
- Write a tester class or a driver class.
- Create an object of a X class.
- Call the function f( ) through X class object reference.
- What will happen ?

Guess the Output:

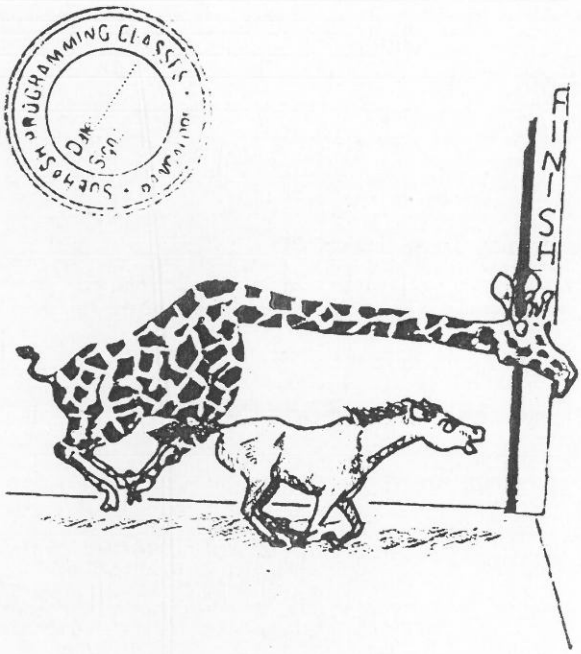


1.

```
abstract class Einstein
{
    abstract public void teach( );
}

abstract class Subhash extends Einstein
{
    abstract public void mentor( );
}

class Anil extends Subhash
{

    public void mentor( )
    {
    }

    public void program( )
    {
        System.out.println( "Programmer" );
    }
}
```

```
public class InheritanceTest
{
    public static void main( String [] args )
    {
        Anil a = new Anil();
        a.program( );
    }
}
```

Polymorphism Advantages:



1.

```
abstract class Animal
{
    abstract public void Display( );
}

class Dog extends Animal
{
    public void Display( )
    {

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

class Cat extends Animal
{
    public void Display( )
    {
        System.out.println( "Cat" );
    }
}

class Goat extends Animal
{
    public void Display( )
    {
        System.out.println( "Goat" );
    }
}

public class PolymorphismAdvantage_1
{
    public static void main( String [] args )
    {
        Dog d = new Dog();
        Cat c = new Cat();
        Goat g = new Goat();
```

```
        CallHim(d);
        CallHim(c);
        CallHim(g);
    }

    public static void CallHim( Animal a )
    {
        a.Display();
    }
}
```

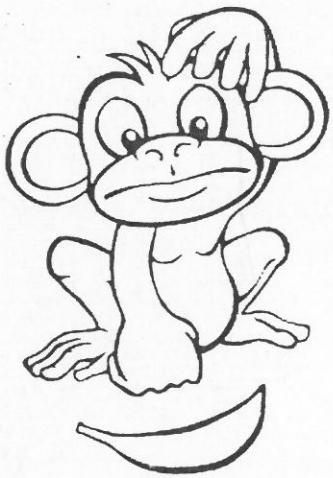
2.

```
abstract class Teacher
{
    abstract public void teach( );
}

class Pteacher extends Teacher
{
    public void teach( )
    {
        System.out.println( "Teaches Physics" );
    }
}
```

```
class Cteacher extends Teacher
{
    public void teach( )
    {
        System.out.println( "Teaches Chemistry" );
    }
}

class Mteacher extends Teacher
{
    public void teach( )
    {
        System.out.println( "Teaches Mathematics" );
    }
}

class MyOwnList
{
    private Teacher [] t = new Teacher [5];
    private int count = 0;

    public void add( Teacher a )
```

```
    {
        if( count < t.length )
        {
            t[count] = a;
            System.out.println( "Teacher added at location " + count );
            count++;
        }
    }
}
public class PolymorphismAdvantage_2
{
    public static void main( String [] args )
    {
        Pteacher p = new Pteacher();
        Cteacher c = new Cteacher();
        Mteacher m = new Mteacher();

        MyOwnList list = new MyOwnList();
        list.add(p);
        list.add(c);
        list.add(m);
    }
}
```

Guess the Output:



1.

```
abstract class Teacher
{
}
class Pteacher extends Teacher
{
}

public class EndingPolymorphism_1
{
    public static void main(String [] args)
    {
        Pteacher t = new Pteacher( );
        System.out.println( t.hashCode());

        Object o = CallHim( t );
        t = o;
        System.out.println( t.hashCode());
        }
    }

    public static Object CallHim( Object o )
```

```
    {
        return o;
    }

}
```

2.

```
abstract class Teacher
{
}
class Pteacher extends Teacher
{
}

public class EndingPolymorphism_2
{
    public static void main(String [ ] args)
    {
        Pteacher t = new Pteacher( );

        System.out.println( t.hashCode( ));

        Object o = CallHim( t );
        t = (Pteacher) o;
        System.out.println( t.hashCode());
    }

    public static Object CallHim( Object o )
    {
        return o;
    }
}
```

3.

```
abstract class Teacher
{
}
class Pteacher extends Teacher
{
}
class Cteacher extends Teacher
{
}

public class EndingPolymorphism_3
{
    public static void main(String [] args)
```

```
    {
        Pteacher t = new Pteacher( );
        Cteacher c = new Cteacher( );

        System.out.println( t.hashCode( ));
        System.out.println( c.hashCode( ));

        Object o = CallHim( c );
        t = (Pteacher) o;
        System.out.println( t.hashCode());
    }

    public static Object CallHim( Object o )
    {
        return o;
    }
}
```

4.

```
abstract class AAA
{
    AAA( )
    {
        System.out.println( "Hello" );
    }
}

class BBB extends AAA
{
    BBB( )
    {
        System.out.println( "Hi" );
    }
}

public class AbstractTesting
{
    public static void main( String [] args )
    {
        AAA a = new BBB( );
    }
}
```

5.

```
class AAA
{
    AAA( )
```

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

abstract class BBB extends AAA
{
    BBB( )
    {
        System.out.println( "Hi" );
    }
}

public class AbstractTesting
{
    public static void main( String [] args )
    {
        AAA a = new BBB( );
    }
}
```

'instanceof' operator:

```
abstract class Teacher
{
}

class Pteacher extends Teacher
{
}

public class EndingPolymorphism_2
{
    public static void main(String [] args)
    {
        Pteacher t = new Pteacher( );
        System.out.println( t.hashCode());

        Object o = CallHim( t );
        if( o instanceof Pteacher ) //instanceof operator
        {
            t = (Pteacher)o;
            System.out.println( t.hashCode());
        }
    }

    public static Object CallHim( Object o )
    {
```

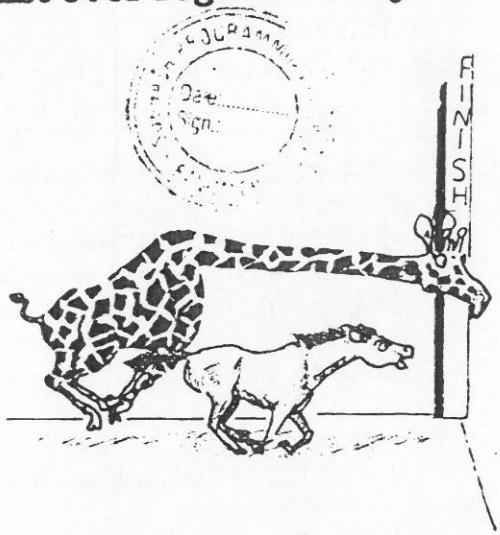
```
        return o;
    }

}
```

The wonderful 'ArrayList':

```
import java.util.*;
class Elephant
{
}

public class ArrayListTest
{
    public static void main( String [] args )
    {
        ArrayList <Elephant> myList = new ArrayList<Elephant>();

        Elephant a = new Elephant( );
        Elephant b = new Elephant( );
        Elephant c = new Elephant( );

        myList.add(a);
        System.out.println(myList.size( ));

        myList.add(b);
        System.out.println(myList.size( ));

        System.out.println(myList.contains(c));
        System.out.println(myList.contains(a));
        System.out.println(myList.indexOf(b));
        System.out.println(myList.get(0));
        System.out.println(myList.get(1));

        myList.remove(0);
        System.out.println( myList.size( ));

        boolean isThere = myList.isEmpty( );
        System.out.println(isThere);

        myList.remove(0);

        isThere = myList.isEmpty( );
        System.out.println(isThere);

        System.out.println(myList.size( ));

        myList.add(c);
```

```
        System.out.println(myList.size( ));
        myList.remove(c);
        System.out.println(myList.size( ));
    }
}
$ java ArrayListTest
1
2
false
true
1
Elephant@6d06d69c
Elephant@7852e922
1
false
true
0
1
0
$
```

What is the advantage of ArrayList over regular Arrays ?

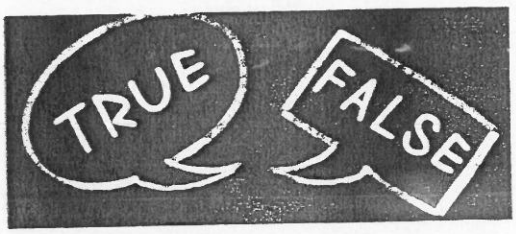


1. String [] s = new String [3];
1. ArrayList <Elephant> s = new ArrayList <Elephant>( );

2. s[2] = "Subhash";
2. s.add(a);

3. Try to remove an element from the middle of an array.
3. How easy it is in an ArrayList.

State True or False :



1. '**final**' variables cannot be modified :
2. 'static' methods are overridden :
3. 'abstract' class can have constructors defined :
4. Non-abstract classes are called concrete classes :

5. We can create an object of an abstract class :
6. Concrete classes can contain abstract methods :
7. Both the overriding method and the overridden method should match in return type and signature :
8. If the overridden method in the base class is public, the overridden method in the child class can be private :

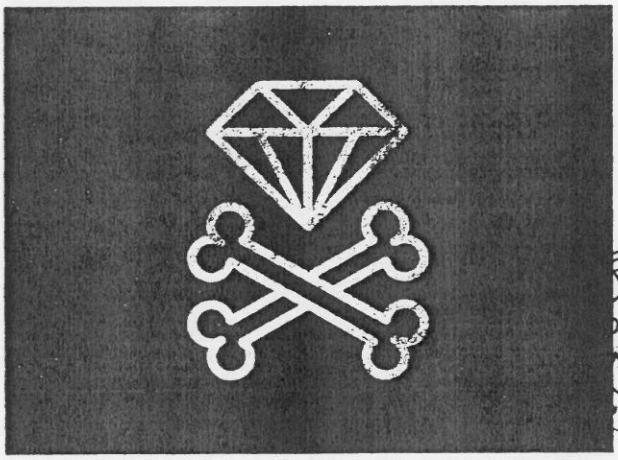
Fill in the blanks :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

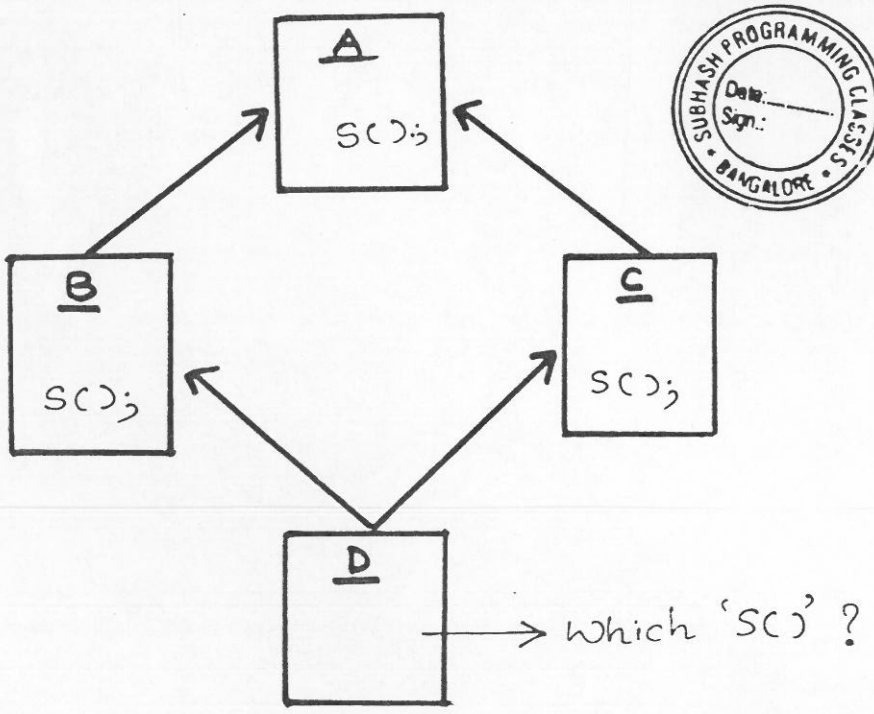
A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

1. Non abstract classes are called as _____ .
2. If the child class does not override the abstract method of the base class, the child class becomes an _____ .
3. The method to add an object into ArrayList is _____ .
4. The method to find the unique identity number of an object is _____ .
5. The father of all classes is the _____ class.
6. Base class reference pointing to subclass object is known as _____ .
7. The actual object being referred by a reference variable is determined through _____ operator.
8. To avoid class inheritance, we need to mark the class as _____ .

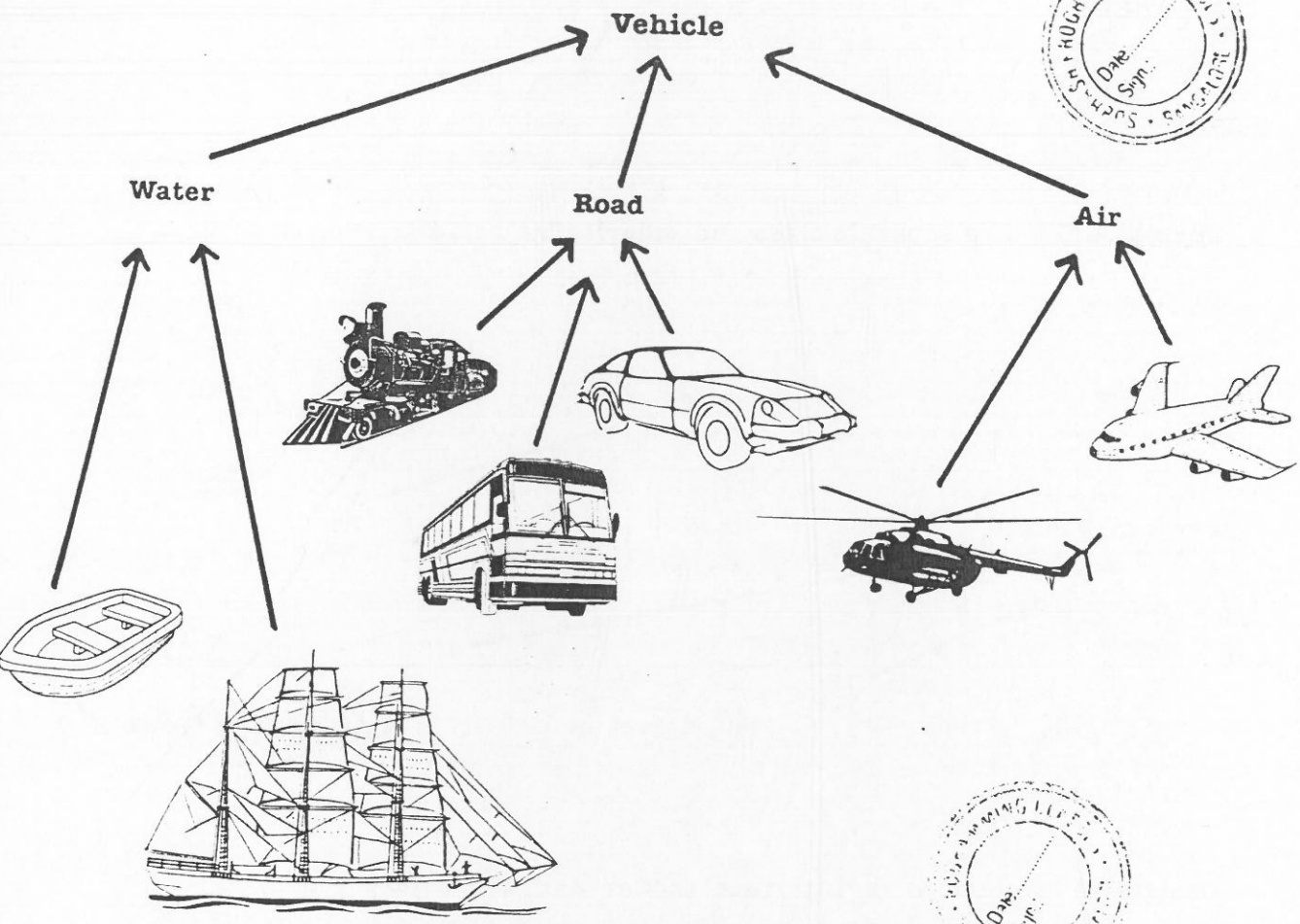
Preventing "Deadly Diamond of death" using 'Interfaces'



'Diamond-Case' problem with multiple inheritance:



Add 'ForPersonalUse( )' method to the design:



Design - 1: Add it to the 'Vehicle' class.

Problems:

Design - 2: Add it to the 'Vehicle' class and make it 'abstract'.

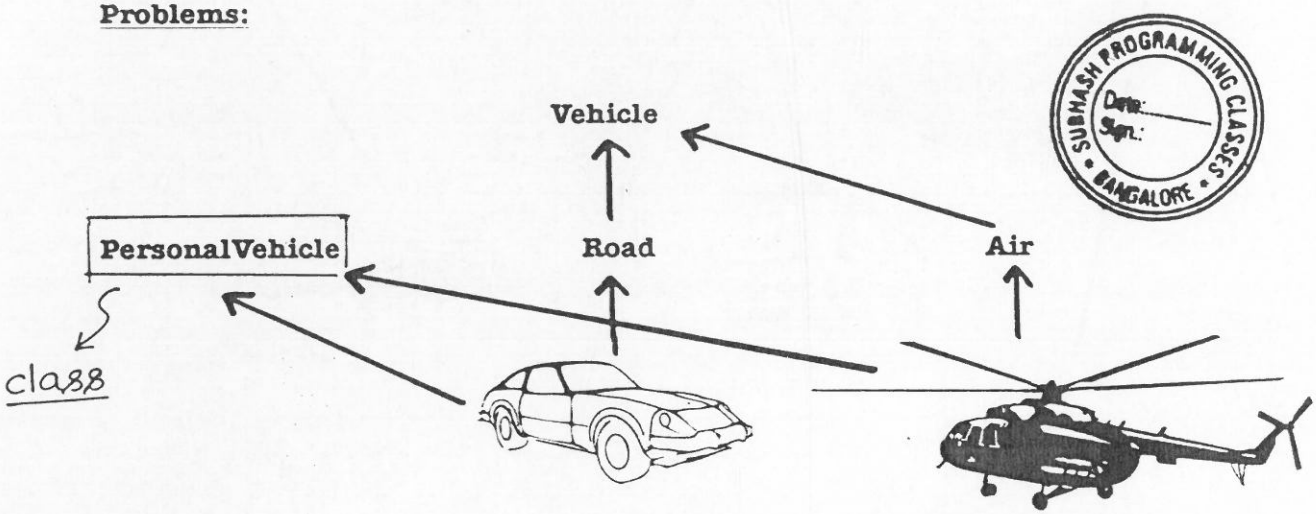
Problems:

Design - 3: Add it only to the class that needs it. Say 'Car' and 'Helicopter'.

Problems:

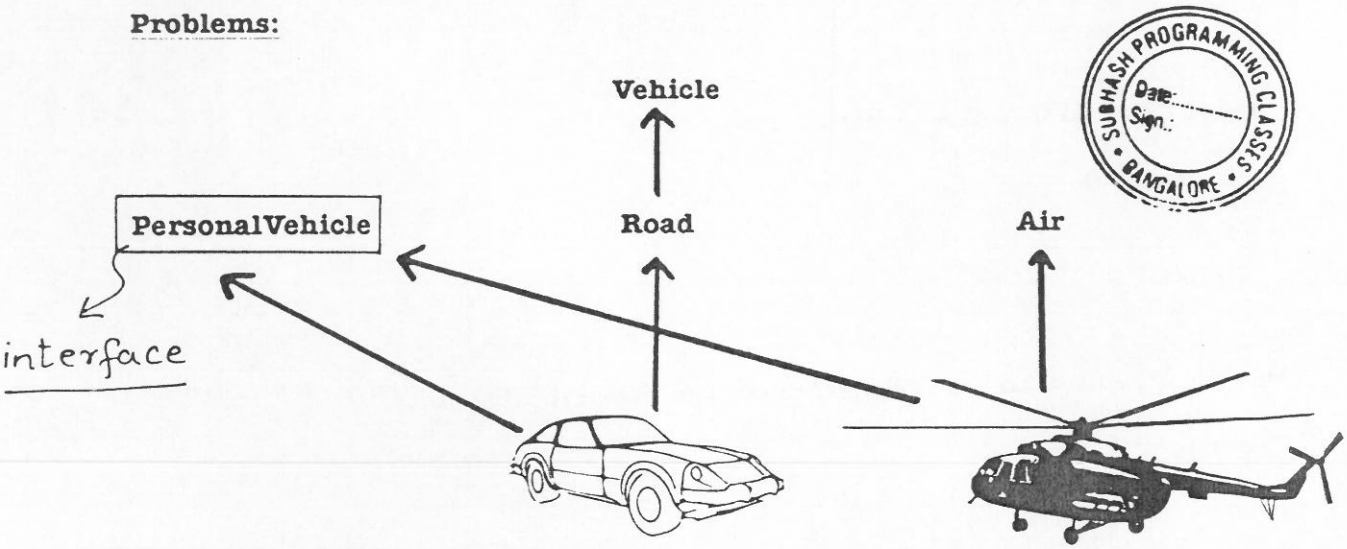
Design - 4: Write a separate class and inherit 'Car' and 'Helicopter' from it.

Problems:



Design - 4: Implement an 'interface' in 'Car' and 'Helicopter'.

Problems:



- "An interface is an 100% abstract class"
- An interface is a class-like construct that contains only constants and abstract methods.
- An interface can be used to define common behaviour for classes (including unrelated classes)
- In many ways an interface is similar to an abstract class, but its intent is to specify common behaviour for objects of related classes or unrelated classes

'interface' general form:

```
public interface interface_name
{
    //constants & method declarations
}
class WhoEverImplementsInterface implements interface_name
{
    //override the interface methods.
}
```

'interface' Example:

```
public interface Edible
{
    public String howToEat( );
}
abstract class Animal
{
    public abstract String sound( );
}

class Chicken extends Animal implements Edible
{
    public String howToEat( )
    {
        return "Chicken: Fry it";
    }

    public String sound( )
    {
        return "Chicken: cock-a-doodle-doo";
    }
}

class Tiger extends Animal
{
    public String sound( )
    {
```

```
            return "Tiger: ROAR";
        }
}

abstract class Fruit implements Edible
{

}

class Apple extends Fruit
{
        public String howToEat( )
        {
            return "Apple: Make apple juice";
        }
}

class Orange extends Fruit
{
        public String howToEat( )
        {
            return "Orange: Make orange juice";
        }
}

public class TestEdible
{
        public static void main( String [] args )
        {
            Object [] objects = { new Tiger( ), new Chicken( ), new Apple( ) };
            for( int i = 0; i < objects.length; i++ )
            {
                if( objects[i] instanceof Edible )
                {
                    System.out.println(((Edible)objects[i]).howToEat( ));
                }
                if( objects[i] instanceof Animal )
                {
                    System.out.println( ((Animal)objects[i]).sound( ));
                }
            }
        }
}
```

Output:

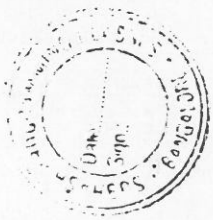
```
Tiger: ROAR
Chicken: Fry it
Chicken: cock-a-doodle-doo
Apple: Make apple Juice
```

Guess the Output:

1.

```
interface IFace
{
    void interface_function( );
}

class IFaceTest implements IFace
{
    public void interface_function( )
    {
        System.out.println( "Interface Method Implementation" );
    }

    void someother_function( )
    {
        System.out.println( "Some other function implementation" );
    }
}

public class InterfaceDemo1
{
    public static void main( String [] args )
    {
        IFace inf = new IFaceTest( );
        inf.interface_function( );
    }
}
```

2.

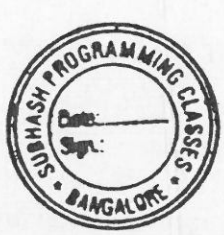


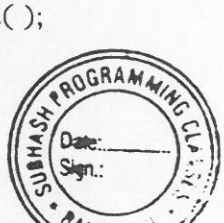
```
interface IFace
{
    void interface_function( );
}

class IFaceTest implements IFace
{
    public void interface_function( )
    {
        System.out.println( "Interface Method Implementation" );
    }

    public void someother_function( )
    {
        System.out.println( "Some other function implementation" );
    }
```

```
}

public class InterfaceDemo2
{
    public static void main( String [] args )
    {
        IFace inf = new IFaceTest( );
        inf.someother_function( );
    }
}
```

3.

```
interface IFace
{
    void interface_function( );
}

class IFaceTest implements IFace
{
    public void interface_function( )
    {
        System.out.println( "Interface Method Implementation" );
    }

    public void someother_function( )
    {
        System.out.println( "Some other function implementation" );
    }
}

public class InterfaceDemo3
{
    public static void main( String [] args )
    {
        IFaceTest inf = new IFaceTest( );
        inf.interface_function( );
        inf.someother_function( );
    }
}
```





4.

```
interface IFace
{
    void interface_function( );
}

class IFaceTest1 implements IFace
```

```
{
    public void interface_function( )
    {
        System.out.println( "Interface1 Method Implementation" );
    }
}

class IFaceTest2 implements IFace
{
    public void interface_function( )
    {
        System.out.println( "Interface2 Method Implementation" );
    }
}

public class InterfaceDemo4
{
    public static void main( String [] args )
    {
        IFace if1 = new IFaceTest1( );
        if1.interface_function( );

        if1 = new IFaceTest2( );
        if1.interface_function( );
    }
}
```

5.


```
interface IFace
{
    void interface_function( );
}

abstract class IFaceTest1 implements IFace
{
}

class IFaceTestDerived extends IFaceTest1
{
    public void interface_function( )
    {
        System.out.println( "In derived function" );
    }
}

public class InterfaceDemo5
{
```

```
    public static void main( String [] args )
    {
        IFace infl = new IFaceTestDerived( );
        infl.interface_function( );
    }
}
```

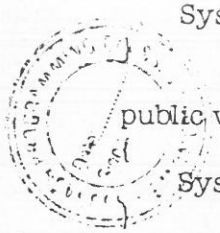
6.


```
class NestedInterface
{
    interface NIF
    {
        void NIFFunction( );
    }
}

class IFaceTest implements NestedInterface.NIF
{
    public void NIFFunction( )
    {
        System.out.println( "NIFFunction implementation" );
    }
}

public class InterfaceDemo6
{
    public static void main( String [] args )
    {
        NestedInterface.NIF nif = new IFaceTest( );
        nif.NIFFunction( );
    }
}
```

7.


```
interface Base
{
    void Function1( );
    void Function2( );
}

interface Derived extends Base
{
    void Function3( );
}

class IFTest implements Derived
{
```

```
    public void Function1( )
    {
        System.out.println( "Function-1" );
    }

    public void Function2( )
    {
        System.out.println( "Function-2" );
    }

    public void Function3( )
    {
        System.out.println( "Function-3" );
    }
}

public class InterfaceDemo7
{
    public static void main( String [] args )
    {
        IFTest ift = new IFTest( );
        ift.Function1( );
        ift.Function2( );
        ift.Function3( );
    }
}
```



'Comparable' interface:

- defines '**compareTo**' method for comparing objects to find the larger of 2 dates, 2 circles, 2 rectangles or 2 squares etc.

Generic Form:

```
public interface Comparable <E>
{
    public int compareTo( E o );
}
```

Standard Examples that implements 'Comparable' interface:

```
public class ComparingStandardClasses
{
    public static void main( String [] args )
    {
        System.out.println( "Subu".compareTo("SUBU"));
        System.out.println( new Integer(5).compareTo(new Integer(6)));
        System.out.println( new Integer(5).compareTo(new Integer(5)));
        java.util.Date date1 = new java.util.Date(2012, 1, 1);
```

```
        java.util.Date date2 = new java.util.Date(2013, 2, 1);

        System.out.println( date1.compareTo(date2));
    }
}
```

Output:

```
$ java ComparingStandardClasses
32
-1
0
-1
$
```

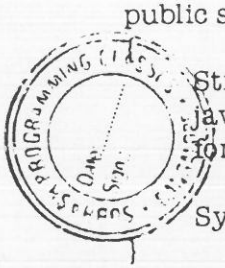
'Comparable' interface would be implemented by standard wrapper classes as follows:

```
public class Integer extends Number implements Comparable <Integer>
{
    public int compareTo(Integer o)
    {
        // ..... implementation
    }
}

public class String extends Object implements Comparable <String>
{
    public int compareTo(String o)
    {
        // ..... implementation
    }
}

public class Date extends Object implements Comparable <Date>
{
    public int compareTo(Date o)
    {
        // ..... implementation
    }
}
```

Important Note:

"Since all **Comparable** objects have the **compareTo** method, the **java.util.Arrays.sort(Object [ ])** method in the Java API uses the **compareTo** method to compare and sorts the objects in an array, provided that the objects are instance of the **Comparable** interface.

```
public class SortComparableObjects
{
    public static void main( String [] args )
    {
        String [ ] cities = { "Mumbai", "Bangalore", "Chennai" };
        java.util.Arrays.sort(cities);
        for( String city: cities )
            System.out.println( city + " " );
        System.out.println();
    }
}
```



```
$ java SortComparableObjects
Bangalore
Chennai
Mumbai
$

class Monkey implements Comparable<Monkey>
{
    private int height;

    public Monkey( int h )
    {
        height = h;
    }

    public int compareTo( Monkey o)
    {
        if( height > o.height )
            return 1;
        else if( height < o.height )
            return -1;
        else
            return 0;
    }
    public String toString( )
    {
        return super.toString() + " " + height ;
    }
}

public class MonkeyTest
{
    public static void main( String [] args )
    {
        Monkey [] monkeys = { new Monkey(2), new Monkey(5), new Monkey(1), new Monkey(3), new Monkey(4) };
        java.util.Arrays.sort(monkeys);
```

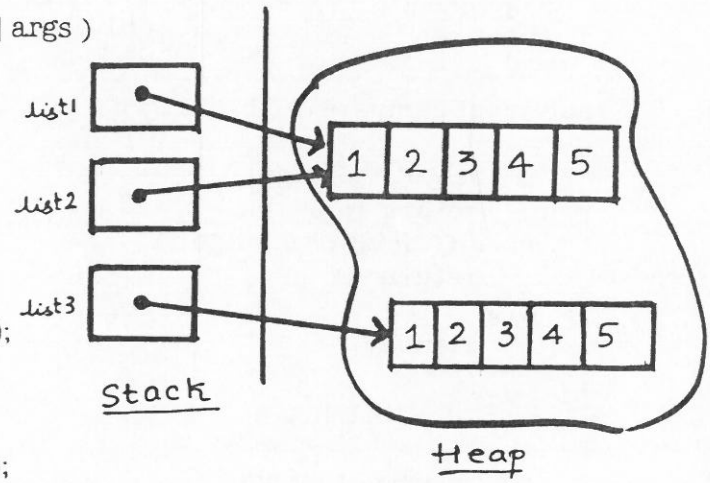
```
        for( Monkey m : monkeys )
        {
            System.out.println( m.toString());
        }
    }
}
```

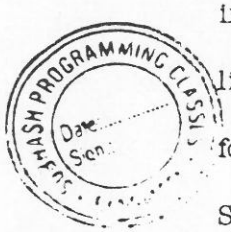
```
$ java MonkeyTest
Monkey@6d06d69c 1
Monkey@7852e922 2
Monkey@4e25154f  3
Monkey@70dea4e   4
Monkey@5c647e05 5
$
```

'Cloneable' interface:

- To create a copy of an object, we need to implement **'Cloneable'** interface.
- **'Cloneable'** interface does not have any method to implement and hence it is called as a 'marker' interface.

```
public class CloningTestWithArray
{
    public static void main( String [] args )
    {
        int [] list1 = { 1, 2, 3, 4, 5 };
        int [] list2 = list1;
        int [] list3 = list1.clone( );

        list2[3] = 9;

        for( int i : list1 )
            System.out.print( i + " " );
        System.out.println();

        for( int i : list2 )
            System.out.print( i + " " );
        System.out.println();

        for( int i : list3 )
            System.out.print( i + " " );
        System.out.println();
    }
}
```



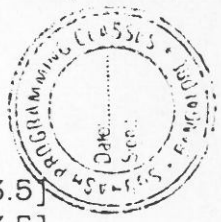


Output:

```
$ java CloningTestWithArray
```

```
1 2 3 9 5
1 2 3 9 5
1 2 3 4 5
$
```

```
public class CloningTestWithArrayList
{
    public static void main( String [] args )
    {
                java.util.ArrayList <Double> list1 = new java.util.ArrayList
<Double>( );
        list1.add(2.3);
        list1.add(3.5);
        list1.add(5.6);
        java.util.ArrayList <Double> list2 = list1;
 java.util.ArrayList <Double> list3 = (java.util.ArrayList<Double>)list1.clone();

        list2.remove(5.6);

        System.out.println( "List1 is " + list1 );
        System.out.println( "List2 is " + list2 );
        System.out.println( "List3 is " + list3 );
    }
}
```

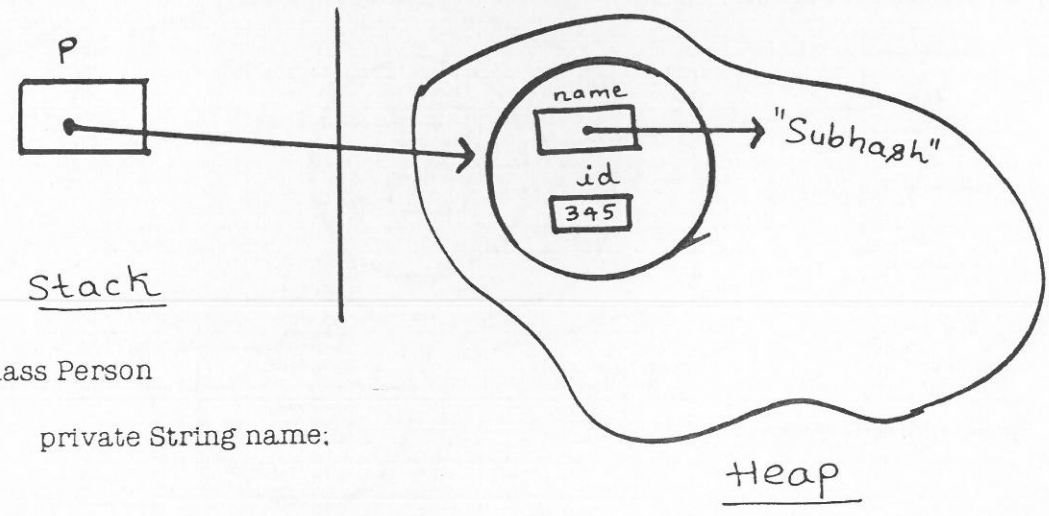


Output:

```
List1 is [2.3, 3.5]
List2 is [2.3, 3.5]
List3 is [2.3, 3.5, 5.6]
```

'Cloning' a Person step-by-step !

Step-1: Introducing a Person. No Cloning



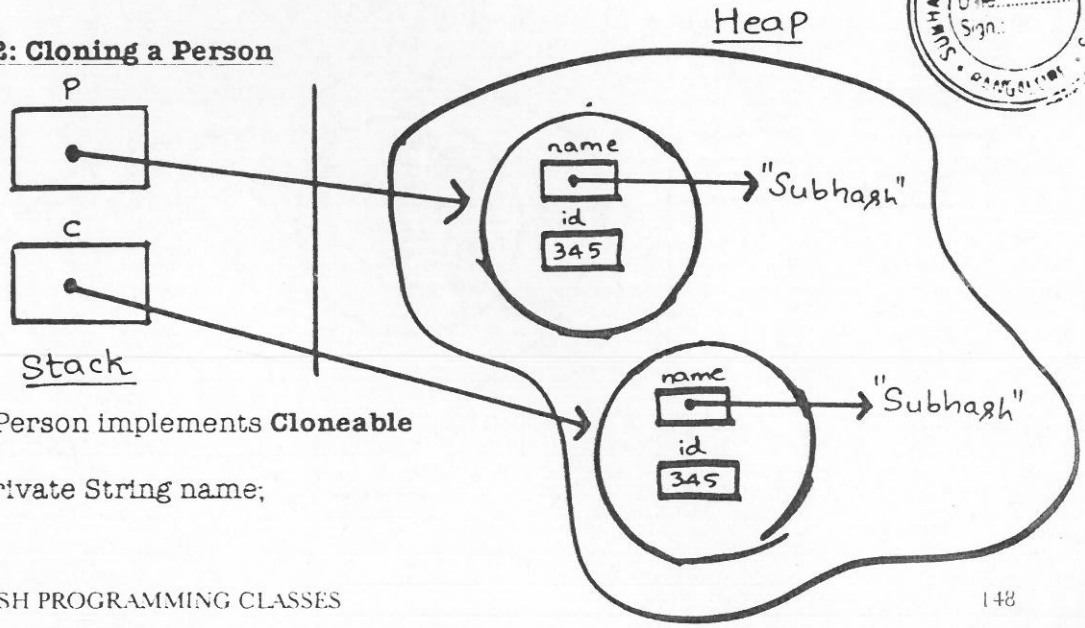

```
class Person
{
    private String name;
```

```
    private int id;

    public Person( String n, int i )
    {
        name = n;
        id = i;
    }

    public String getName()
    {
        return name;
    }
    public void setName( String n)
    {
        name = n;
    }
}

public class PersonCloning
{
    public static void main( String [] args )
    {
        Person p = new Person( "Subhash", 345 );
        System.out.println(p.getName());

    }
}
```

Output:

```
$ java PersonCloning
Subhash
$
```

Step-2: Cloning a Person




```
class Person implements Cloneable
{
    private String name;
```

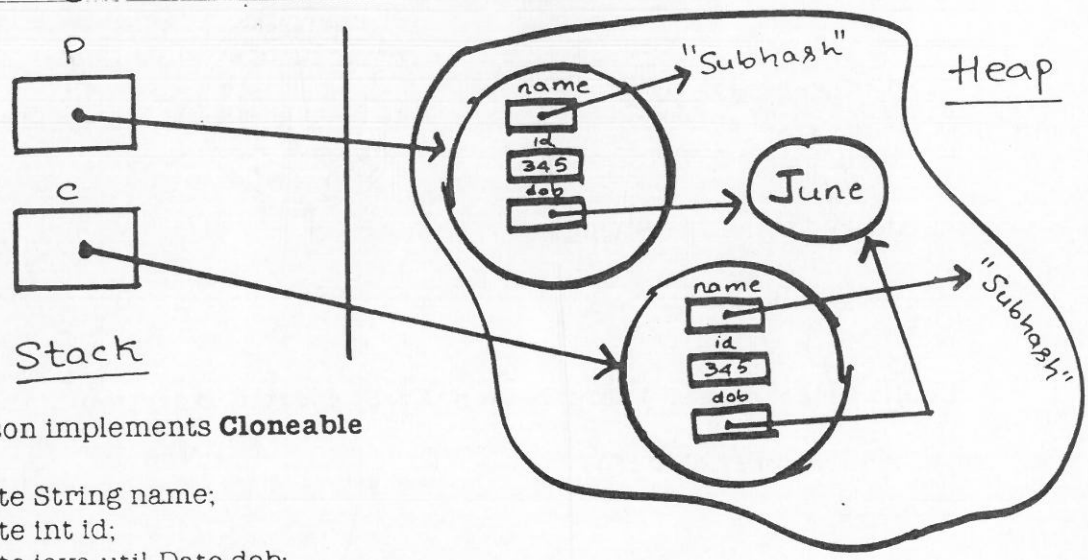
```
    private int id;

    public Person( String n, int i )
    {
        name = n;
        id = i;
    }

    public String getName()
    {
        return name;
    }
    public void setName( String n)
    {
        name = n;
    }

    public Object clone( ) throws CloneNotSupportedException
     {
        return super.clone();
     }
}

public class PersonCloning
{
    public static void main( String [] args )
    {
        Person p = new Person( "Subhash", 345 );
        System.out.println(p.getName());

        try
        {
            Person c = (Person)p.clone( );
            c.setName("Sumesh");
            System.out.println( );
            System.out.println(p.getName());
            System.out.println(c.getName());
        }
        catch ( CloneNotSupportedException e )
        {
            e.printStackTrace();
        }
    }
}
```

Output:

```
$ java PersonCloning
Subhash
```

Subhash
Sumesh
$

Step-3: Cloning a mutable state in a Person - Shallow Copy




```
class Person implements Cloneable
{
    private String name;
    private int id;
    private java.util.Date dob;

    public Person( String n, int i, java.util.Date d )
    {
        name = n;
        id = i;
        dob = d;
    }

    public String getName()
    {
        return name;
    }
    public void setName( String n)
    {
        name = n;
    }

    public Object clone( ) throws CloneNotSupportedException
    {
        return super.clone();
    }
    public java.util.Date getDate()
    {
        return dob;
    }
    public void setDate( java.util.Date dd )
```

Ignore this!!

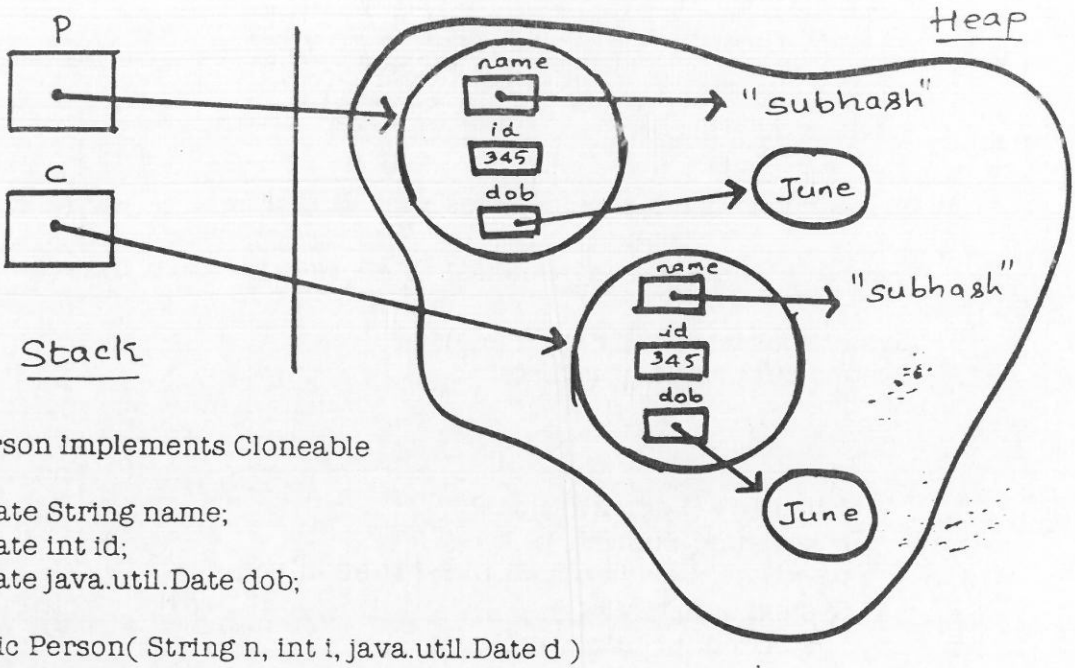
```
    {
        dob.setMonth(2);
    }
}

public class PersonCloning
{
    public static void main( String [] args )
    {
        Person p = new Person( "Subhash", 345, new java.util.Date(1985, 5,
6));
        System.out.println(p.getName());
        System.out.println(p.getDate());

        try
        {
            Person c = (Person)p.clone();
            p.setName("Sumesh");
            p.setDate( new java.util.Date(1986,4, 6));
            System.out.println( );
            System.out.println(p.getName());
            System.out.println(p.getDate());
            System.out.println(c.getName());
            System.out.println(c.getDate());

        }
        catch ( CloneNotSupportedException e )
        {
            e.printStackTrace();
        }
    }
}
```


Used for confusing!! Ignore!! :)

Output:

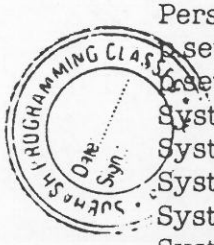
```
$ java PersonCloning
Subhash
Sat Jun 06 00:00:00 IST 3885

Sumesh
Fri Mar 06 00:00:00 IST 3885
Subhash
Fri Mar 06 00:00:00 IST 3885
$
```

Step-4: Cloning a mutable state in a Person - Deep Copy




```
class Person implements Cloneable
{
    private String name;
    private int id;
    private java.util.Date dob;

    public Person( String n, int i, java.util.Date d )
    {
        name = n;
        id = i;
        dob = d;
    }

    public String getName( )
    {
        return name;
    }
    public void setName( String n)
    {
        name = n;
    }

    public Object clone( ) throws CloneNotSupportedException
    {
        Person temp = (Person)super.clone( );
        temp.dob = (java.util.Date)dob.clone( );
        return temp;
    }
    public java.util.Date getDate()
    {
        return dob;
    }
    public void setDate( java.util.Date dd )
    {
        dob.setMonth(2);
```

```
    }
}

public class PersonCloning
{
    public static void main( String [] args )
    {
        Person p = new Person( "Subhash", 345, new java.util.Date(1985, 5,
6));
        System.out.println(p.getName());
        System.out.println(p.getDate());

        try
        {
            Person c = (Person)p.clone();
            p.setName("Sumesh");
            p.setDate( new java.util.Date(1986,4, 6));
            System.out.println();
            System.out.println(p.getName());
            System.out.println(p.getDate());
            System.out.println(c.getName());
            System.out.println(c.getDate());

        }
        catch ( CloneNotSupportedException e )
        {
            e.printStackTrace();
        }

    }
}
```

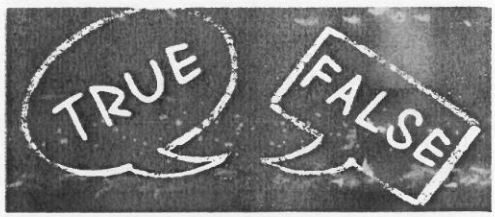


Output:

```
$ java PersonCloning
Subhash
Sat Jun 06 00:00:00 IST 3885

Sumesh
Fri Mar 06 00:00:00 IST 3885
Subhash
Sat Jun 06 00:00:00 IST 3885

$
```

- **Deep copy** of an object will have exact copy of all the fields of original object just like **shallow copy**. But in additional, if original object has any references to other objects as fields, then copy of those objects are also created by calling clone( ) method on them

State True or False :



1. 'interfaces' avoid deadly diamond of death :
2. 2 classes from 2 different class hierarchy can implement 1 interface :
3. 'interfaces' are 100% pure abstract class :
4. 'interfaces' can have instance variables :
5. 'interfaces' cannot be implemented by 2 different classes from the same class hierarchy :
6. 'interfaces' cannot be inherited :
7. 'Integer' class implements 'Comparable' interface.

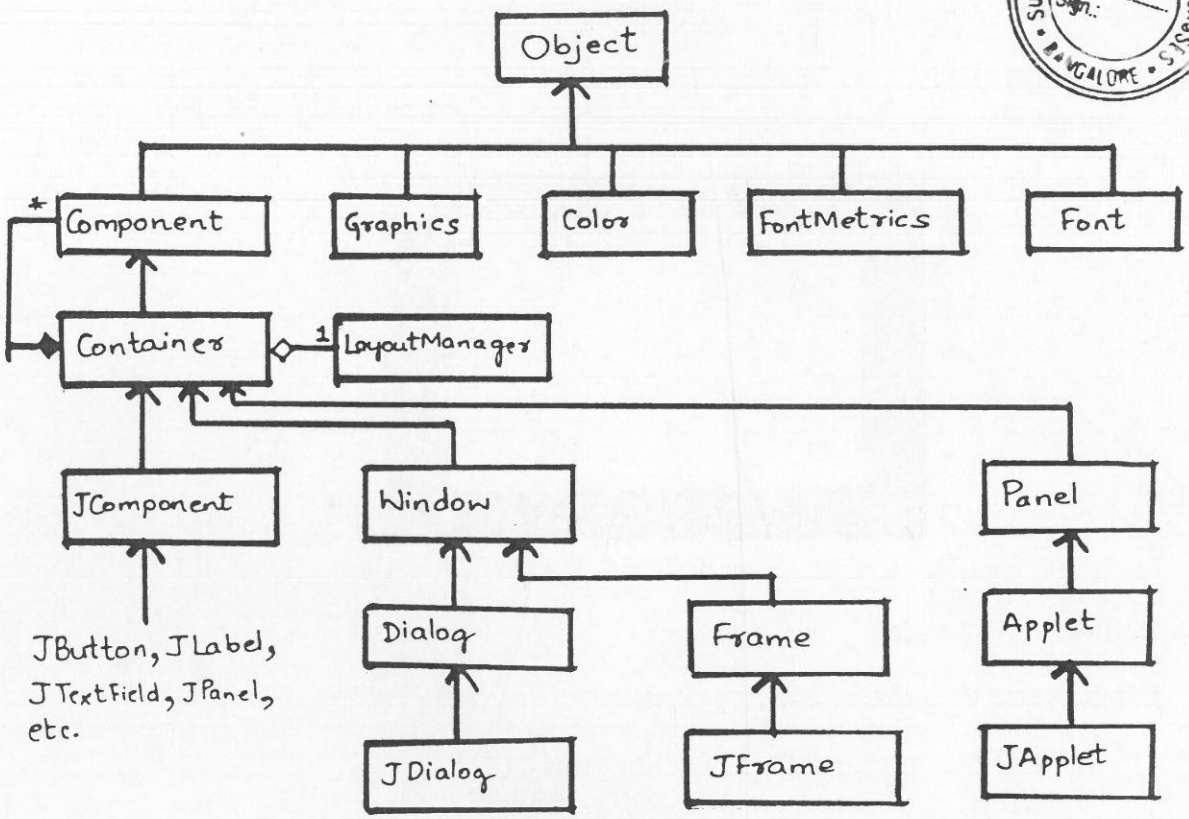
Fill in the blanks :

1. 'Cloneable' interface is known as _____ interface.
2. _____ method has to be overridden if you implement 'Comparable' interface.
3. The two types of making a copy is _____ copy and _____ copy. To avoid _____ copy, we implement _____ copy.
4. Assume cloning is not done. d1 and d2 are 2 different Dog objects. Then, d1.equals(d2) is _____ .
5. Assume cloning is done for a Dog object. d2 is a clone of d1. Then, d1.equals(d2) is _____ .
6. Assume cloning is not done. d1 and d2 are 2 different Dog objects. d1 is assigned to d2. Then, d1.equals(d2) is _____ .

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

GUI Programming



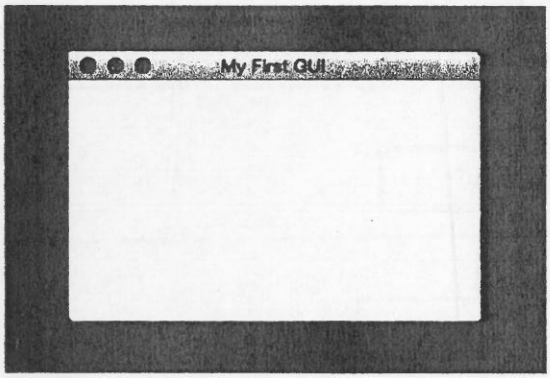
Classification of GUI related classes

- **Container Classes**
  - JFrame, JPanel, JDialog, JApplet etc.
  - Used to hold other components
- **Component Classes**
  - JRadioButton, JTextField, JButton, JComboBox, JLabel etc.
  - Gets stuck on the container.
- **Helper Classes**
  - Font, Color, Dimension, LayoutManager etc.
  - Used for decorating the GUI.

Basic rules for creating a simple GUI:

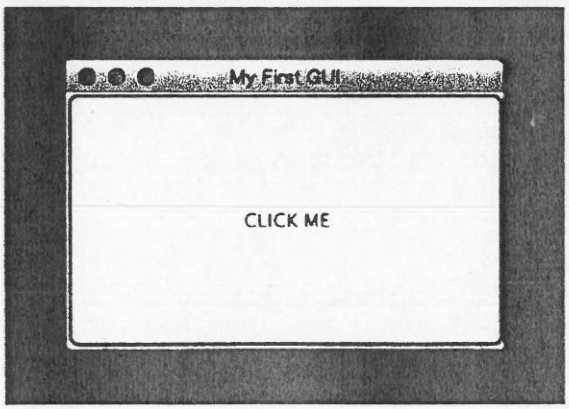
- Create either a FRAME or an APPLET
- Stick PANEL to a FRAME
- Hook up the COMPONENTS

My First GUI !

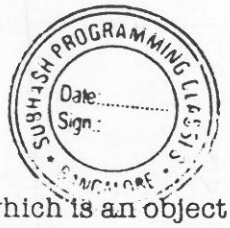



```
import javax.swing.*;
public class MyFirstGui
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "My First GUI" );

        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

My First GUI with a component !



```
import javax.swing.*;
public class MyFirstGuiWithButton
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "My First GUI" );

        JButton button = new JButton( "CLICK ME" );

        frame.add(button);
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

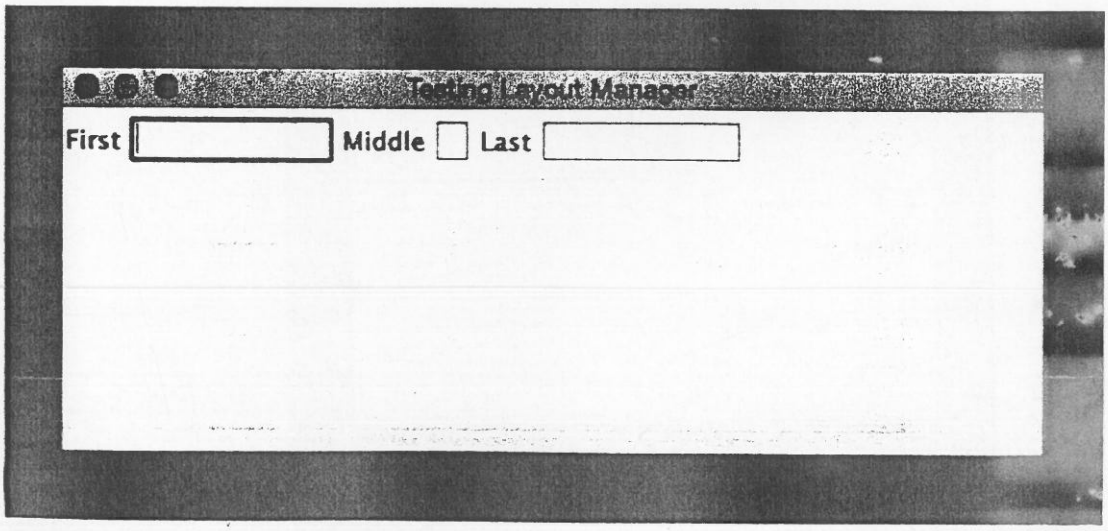


Layout Managers:

- Each container contains a layout manager, which is an object responsible for laying out the GUI components in the container.

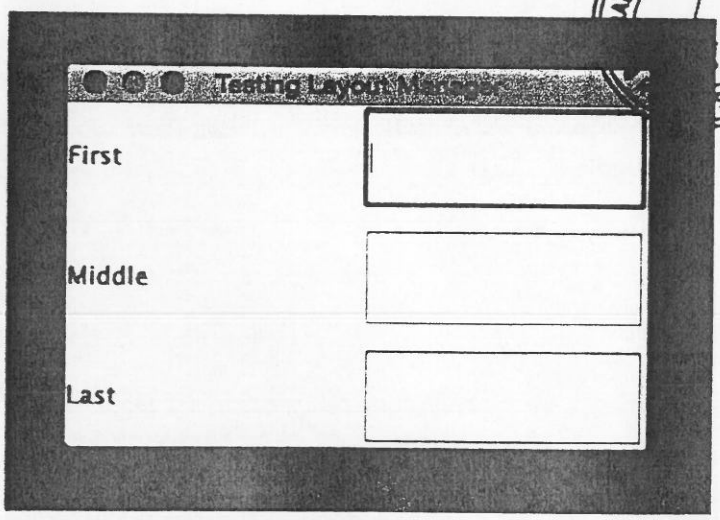
- **FlowLayout**
  - Alignment, horizontal gap, vertical gap
- **GridLayout**
  - Rows, columns, horizontal gap, vertical gap
- **BorderLayout**
  - horizontal gap, vertical gap

FlowLayout Example :



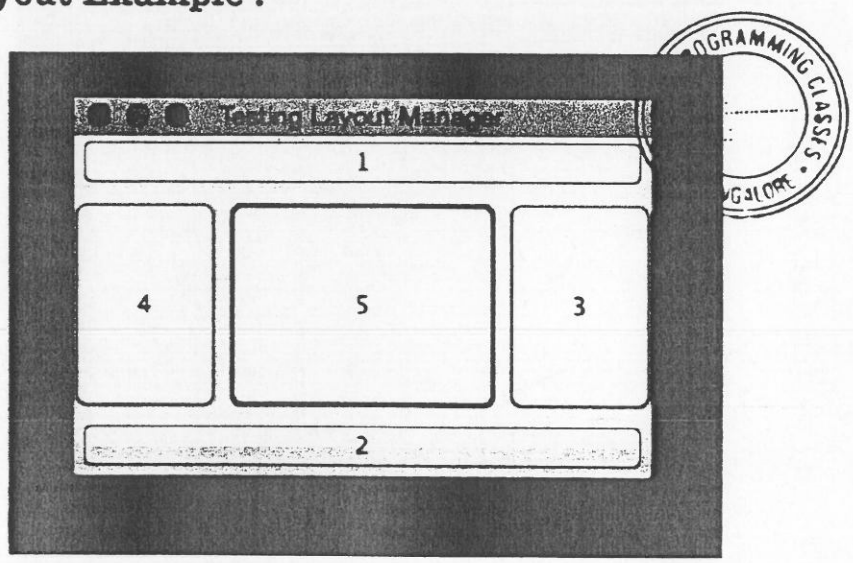
```
import javax.swing.*;
import java.awt.FlowLayout;

public class FlowLayoutTesting
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Testing Layout Manager" );

        JLabel fname = new JLabel( "First" );
        JLabel mname = new JLabel( "Middle");
        JLabel lname = new JLabel( "Last" );

        JTextField ffield = new JTextField(8);
        JTextField mfield = new JTextField(1);
        JTextField lfield = new JTextField(8);

        frame.add(fname);
        frame.add(ffield);
        frame.add(mname);
        frame.add(mfield);
        frame.add(lname);
        frame.add(lfield);

        FlowLayout flayout = new FlowLayout(FlowLayout.LEFT, 3, 3);
        frame.setLayout(flayout);
        frame.setSize(500, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

GridLayout Example :



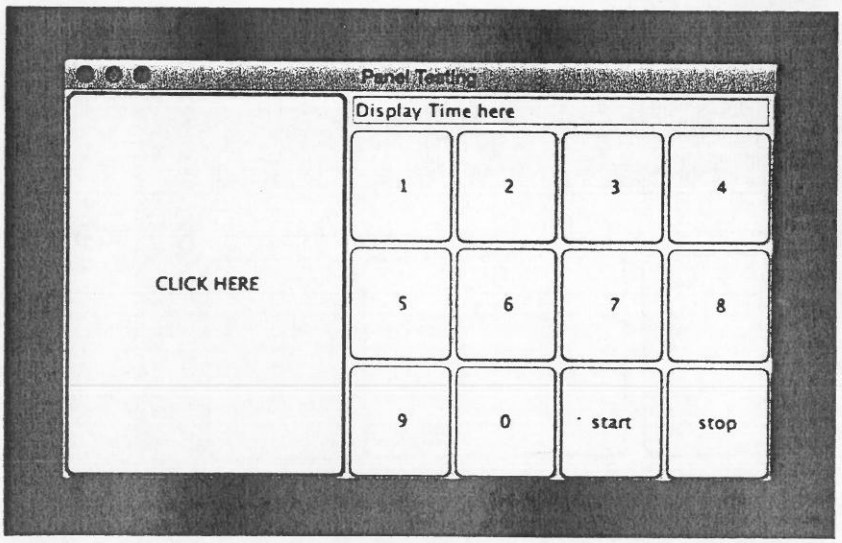
```
import javax.swing.*;
import java.awt.GridLayout;

public class GridLayoutTesting
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Testing Layout Manager" );

        JLabel fname = new JLabel( "First" );
        JLabel mname = new JLabel( "Middle");
        JLabel lname = new JLabel( "Last" );

        JTextField ffield = new JTextField(8);
        JTextField mfield = new JTextField(1);
        JTextField lfield = new JTextField(8);

        frame.add(fname);
        frame.add(ffield);
        frame.add(mname);
        frame.add(mfield);
        frame.add(lname);
        frame.add(lfield);

        GridLayout glayout = new GridLayout(3, 2, 3, 8);
        frame.setLayout(glayout);
        frame.setSize(300, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

BorderLayout Example :

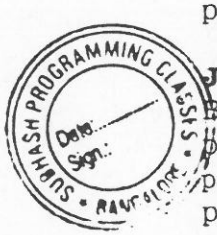


```
import javax.swing.*;
import java.awt.BorderLayout;

public class BorderLayoutTesting
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Testing Layout Manager" );

        JButton button_1 = new JButton("1");
        JButton button_2 = new JButton("2");
        JButton button_3 = new JButton("3");
        JButton button_4 = new JButton("4");
        JButton button_5 = new JButton("5");

        BorderLayout blayout = new BorderLayout(5, 5);
        frame.setLayout(blayout);

        frame.add(button_1, BorderLayout.NORTH);
        frame.add(button_2, BorderLayout.SOUTH);
        frame.add(button_3, BorderLayout.EAST);
        frame.add(button_4, BorderLayout.WEST);
        frame.add(button_5, BorderLayout.CENTER);

        frame.setSize(300, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

Panels are Subcontainers:



```
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.GridLayout;

public class PanelTesting
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame("Panel Testing");

        JPanel panel_1 = new JPanel();
        GridLayout glayout = new GridLayout(3,3,0,0);
        panel_1.setLayout(glayout);
        for( int i = 1; i <= 9; i++ )
        {
            panel_1.add(new JButton(" " + i));
        }
        panel_1.add(new JButton(" 0 " ));
        panel_1.add(new JButton("start"));
        panel_1.add(new JButton("stop"));

        JPanel panel_2 = new JPanel();
        BorderLayout panelLayout = new BorderLayout( );
        panel_2.setLayout(panelLayout);
        panel_2.add(new JTextField("Display Time here"), BorderLayout.NORTH);
        panel_2.add(panel_1);

        BorderLayout frameLayout = new BorderLayout();
        frame.setLayout(frameLayout);
        frame.add( new JButton("CLICK HERE"), BorderLayout.CENTER);
        frame.add(panel_2, BorderLayout.EAST);

        frame.setSize(500, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```



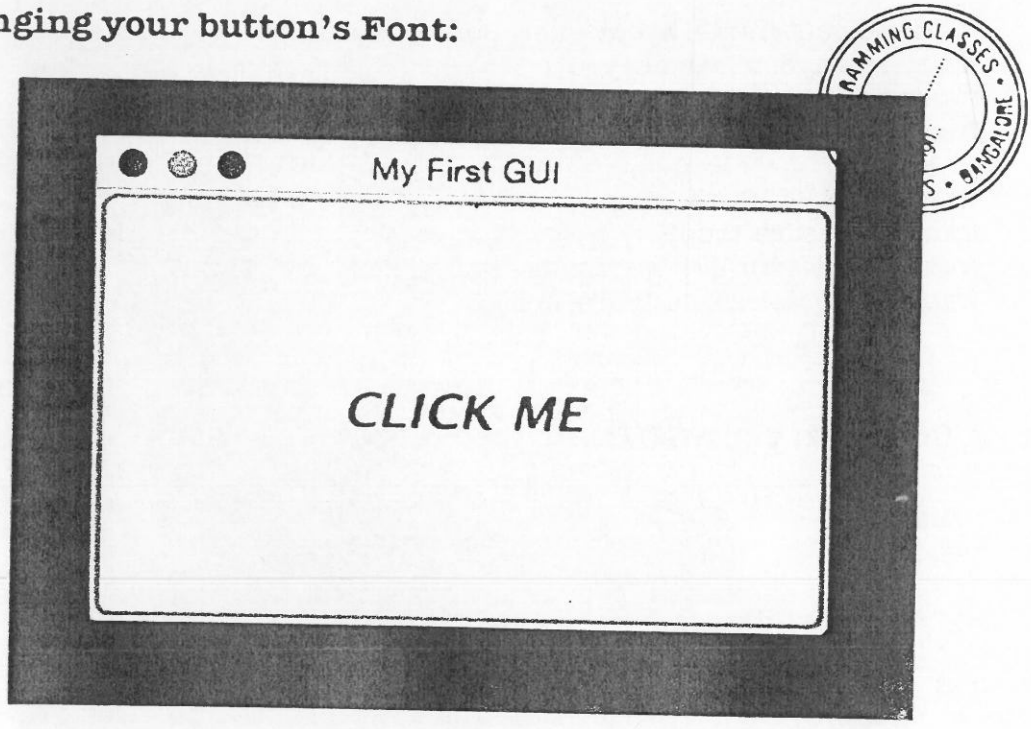
Adding Colors to your GUI:


```
import java.awt.*;
import javax.swing.*;

public class AddColorToFrame
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Colorful Frame" );
```

```
        Color c = new Color( 35, 67, 97 );

        frame.getContentPane().setBackground(c);
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

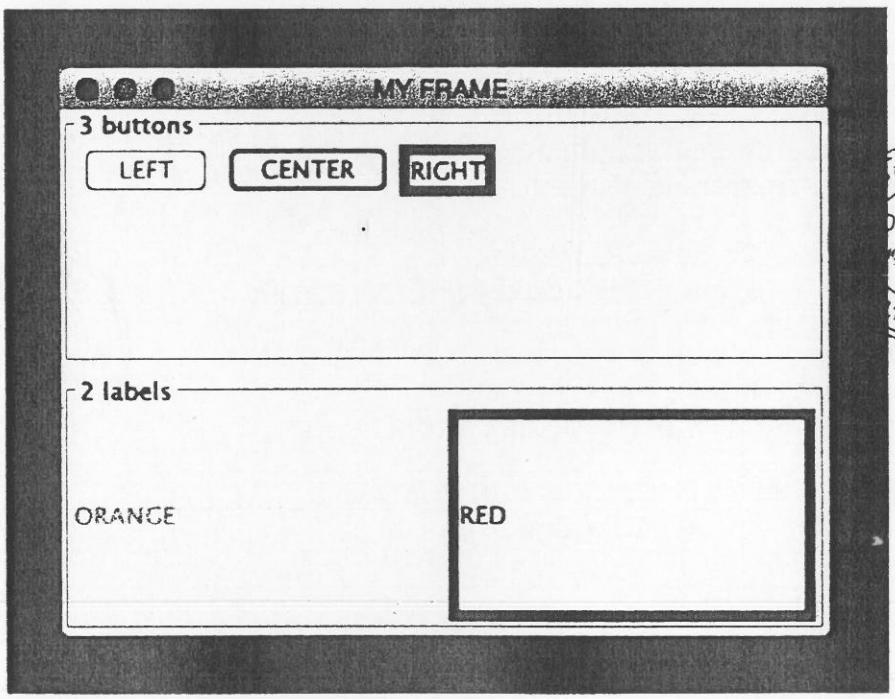
```
import java.awt.*;
import javax.swing.*;

public class AddColorToFrame
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Colorful Frame" );

        frame.getContentPane().setBackground(Color.RED);
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

Changing your button's Font:



```
import javax.swing.*;
import java.awt.*;

public class MyFirstGuiWithButtonFont
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "My First GUI" );

        JButton button = new JButton( "CLICK ME" );
        Font font = new Font( "Quick Sand", Font.ITALIC + Font.BOLD, 20);
        button.setForeground(Color.RED);
        button.setFont(font);

        frame.add(button);
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}
```

Adding Borders:



```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class AddingBorders
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "MY FRAME" );

        JPanel panel_1 = new JPanel();
        FlowLayout flayout = new FlowLayout( FlowLayout.LEFT, 1, 1);
        panel_1.setLayout(flayout);
        JButton bleft = new JButton("LEFT");
        JButton bcenter = new JButton("CENTER");
        JButton bright = new JButton( "RIGHT");

        bleft.setForeground(Color.GREEN);
        bcenter.setForeground(Color.RED);

        bright.setToolTipText("I am here buddy");
        bright.setBorder( new LineBorder(Color.BLACK, 6));

        panel_1.setBorder( new TitledBorder("3 buttons"));
        panel_1.add(bleft);
        panel_1.add(bcenter);
        panel_1.add(bright);
        panel_1.setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));

        JLabel olabel = new JLabel("ORANGE");
        JLabel rlabel = new JLabel("RED");
        rlabel.setBorder(new LineBorder(Color.RED, 6));
        olabel.setForeground(Color.ORANGE);

        JPanel panel_2 = new JPanel();
        GridLayout glayout = new GridLayout(1, 2, 5, 5);
        panel_2.setLayout(glayout);
        panel_2.add(olabel);
        panel_2.add(rlabel);
        panel_2.setBorder( new TitledBorder("2 labels"));

        GridLayout layoutForFrame = new GridLayout( 2, 1, 5, 5 );
        frame.setLayout(layoutForFrame);
        frame.add(panel_1);
        frame.add(panel_2);

        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setLocationRelativeTo(null);
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

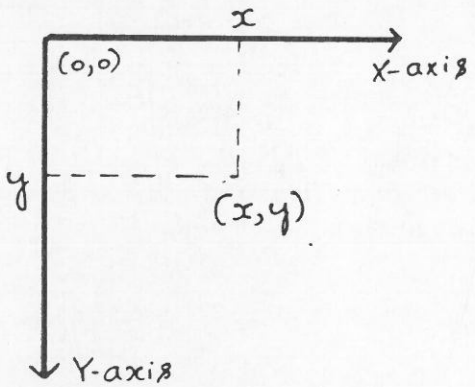
Playing with Icons:


```
import java.awt.*;
import javax.swing.*;

public class PlayingWithIcons
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame();

        ImageIcon spc = new ImageIcon("spc.png");
        ImageIcon sec = new ImageIcon("sec.png");
        ImageIcon muas = new ImageIcon("muas.png");

        JButton button = new JButton("Click Me", muas);
        button.setPressedIcon(spc);
        button.setRolloverIcon(sec);


        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800,500);
        frame.setLocationRelativeTo(null);
        frame.add(button);

    }
}
```

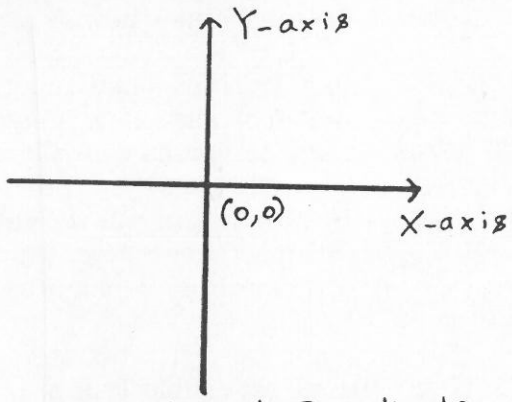
Adding 'Graphics'

- Think of a GUI Component as a piece of paper and the **Graphics** object as a pencil or a paintbrush. You can apply the methods in the **Graphics** class to draw graphics on a GUI component.

- To paint, you need to specify where to paint. Each component has its own co-ordinate system with the origin(0,0) at the upper-left corner. The x-coordinate increases to the right, and the y-coordinate increases downward.

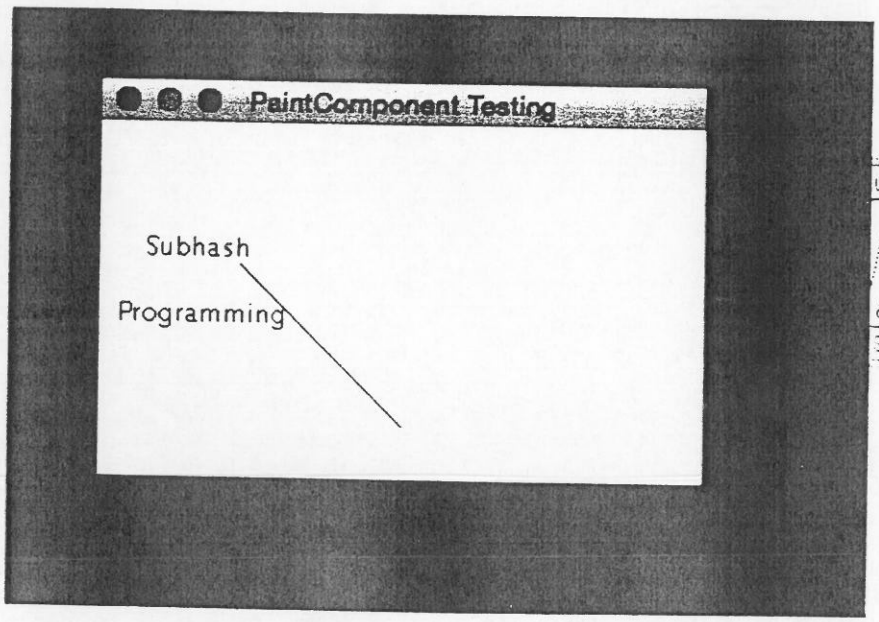


- Java Cordinate System



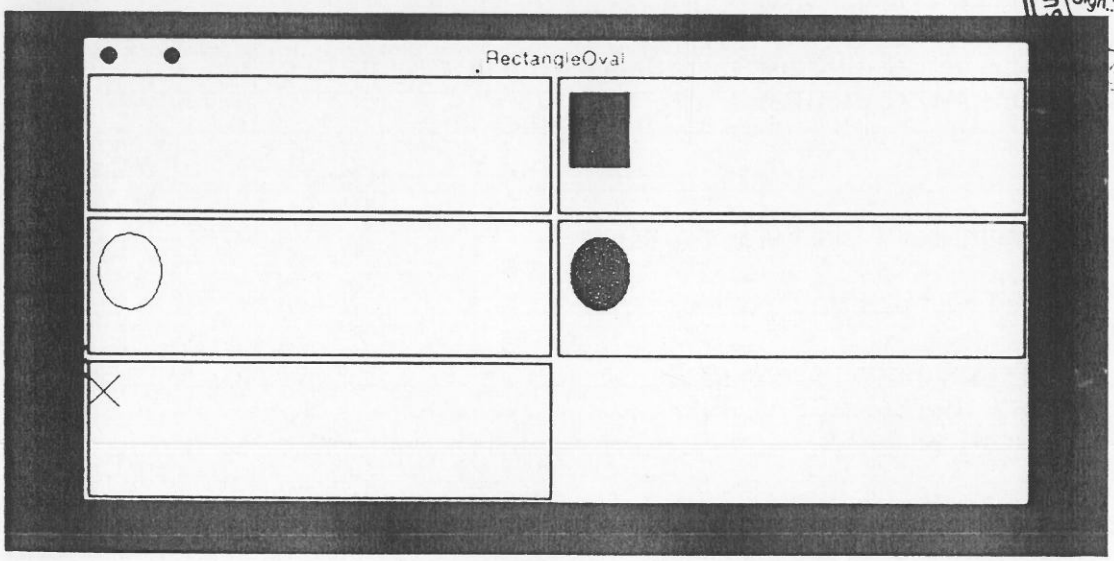
- Conventional Coordinate System

Testing 'paintComponent( )':



```
import java.awt.*;
import javax.swing.*;

public class TestingPaintComponent
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame("PaintComponent Testing");
        PaintComponent pc = new PaintComponent( );
        frame.add(pc);
        frame.setVisible(true);
        frame.setSize(400,300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}

class PaintComponent extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.drawLine(70,70, 50, 50);
        g.drawString("Subhash", 23, 67);
    }
}
```

Adding Rectangles & Ovals:

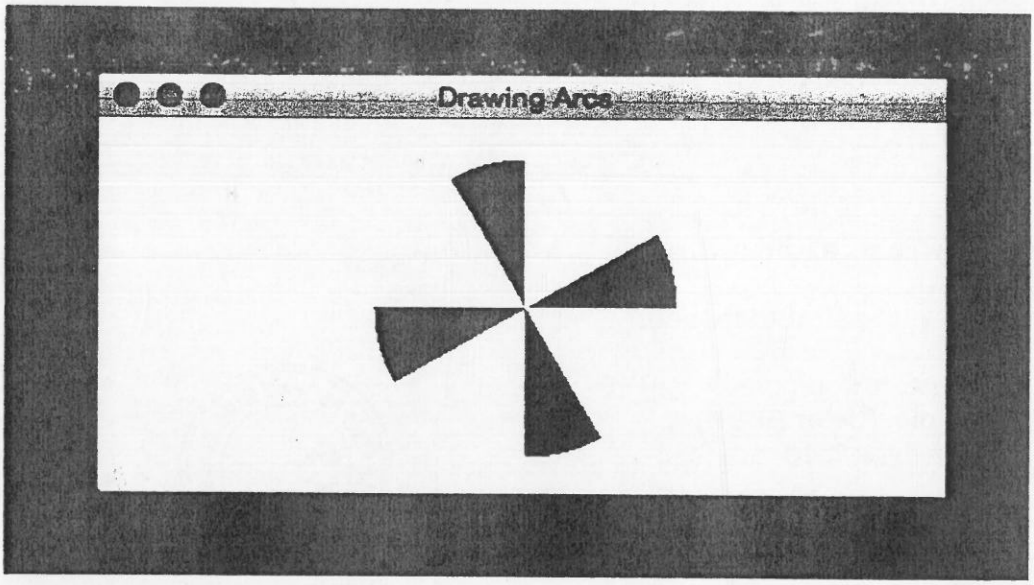



```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class DrawingRectangleOval
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "RectangleOval" );

        RectangleDrawPanel rdp = new RectangleDrawPanel();
        RectangleFillPanel rfp = new RectangleFillPanel();
        OvalDrawPanel      odp = new OvalDrawPanel();
        OvalFillPanel      ofp = new OvalFillPanel();
        CrossLinePanel     clp = new CrossLinePanel();

        rdp.setBorder(new LineBorder(Color.BLACK, 2));
        rfp.setBorder(new LineBorder(Color.BLACK, 2));
        odp.setBorder(new LineBorder(Color.BLACK, 2));
        ofp.setBorder(new LineBorder(Color.BLACK, 2));
        clp.setBorder(new LineBorder(Color.BLACK, 2));

        GridLayout gl = new GridLayout(3,2,3,3);
        frame.setLayout(gl);
        frame.add(rdp);
        frame.add(rfp);
        frame.add(odp);
        frame.add(ofp);
        frame.add(clp);
```


```
        frame.setVisible(true);
        frame.setSize(600, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}

class RectangleDrawPanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.setColor(Color.ORANGE);
        g.drawRect( 7, 10, 40, 50 ); // Try g.drawRoundRect( 7, 10, 40, 50, 15, 20);
    }
}
class RectangleFillPanel extends JPanel
{
```

```
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillRect( 7, 10, 40, 50 ); // Try g.fillRoundRect( 7, 10, 40, 50, 15, 20);
    }
}

class OvalDrawPanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.setColor(Color.BLUE);
        g.drawOval(7, 10, 40, 50);
    }
}
class OvalFillPanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.setColor(Color.GREEN);
        g.fillOval( 7, 10, 40, 50 );
    }
}

class CrossLinePanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        g.drawLine( 0, 10, 20, 30);
        g.drawLine( 20, 10, 0, 30);
    }
}
```

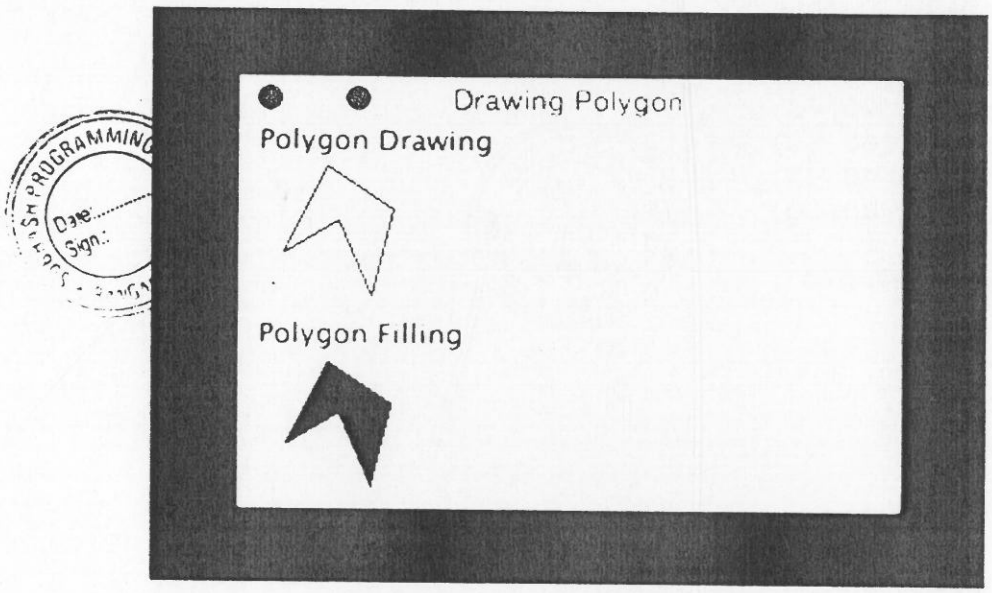
Drawing Arcs:




```
import java.awt.*;
import javax.swing.*;

public class DrawingArcs
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Drawing Arcs" );

        ArcDrawing ad = new ArcDrawing();
        frame.add(ad);
        frame.setVisible(true);
        frame.setSize(400, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}

class ArcDrawing extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);

        int xCenter = getWidth()/2;
        int yCenter = getHeight()/2;
        int radius  = (int)(Math.min(getWidth( ), getHeight()) * 0.4);

        int x = xCenter - radius;
```


```
        int y = yCenter - radius;

        g.setColor(Color.BLUE);
        g.fillArc(x, y, 2 * radius, 2 * radius, 0, 30); // try drawArc( )
        g.fillArc(x, y, 2 * radius, 2 * radius, 90, 30); // try drawArc( )
        g.fillArc(x, y, 2 * radius, 2 * radius, 180, 30); // try drawArc( )
        g.fillArc(x, y, 2 * radius, 2 * radius, 270, 30); // try drawArc( )
    }
}
```

Drawing Polygons:

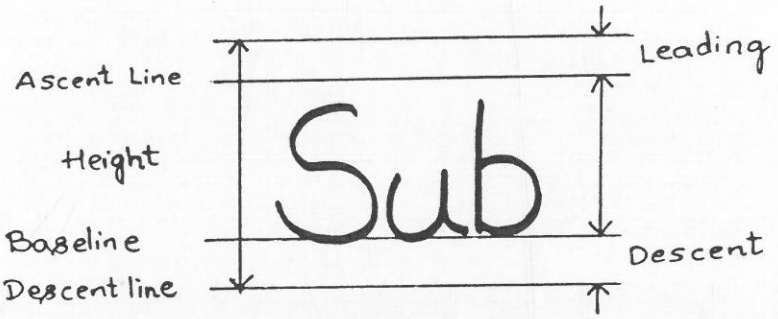


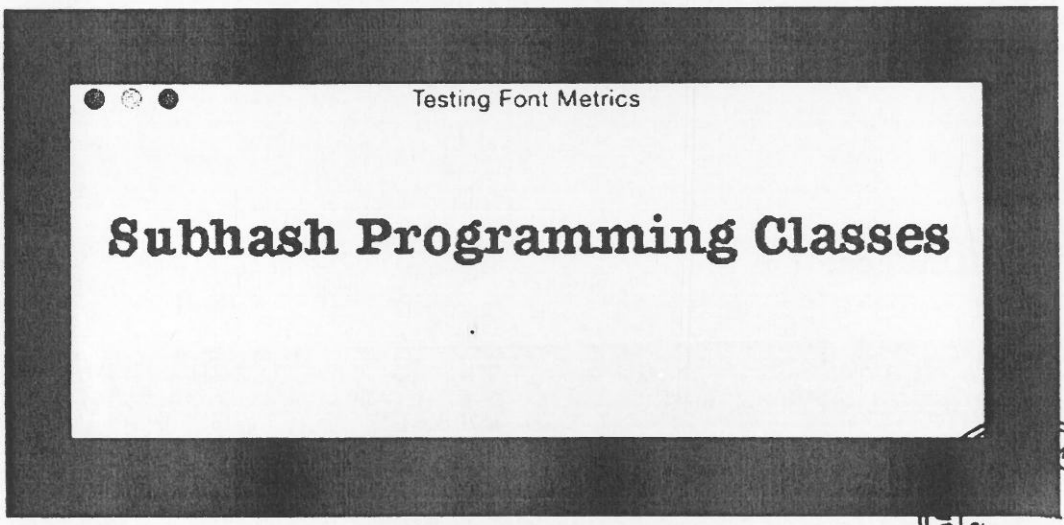

```
import javax.swing.border.*;
import java.awt.*;
import javax.swing.*;

public class DrawPolygon
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame("Drawing Polygon");

        PolygonDrawing pd = new PolygonDrawing();
        pd.setBorder(new TitledBorder("Polygon Drawing"));
        PolygonFilling pf = new PolygonFilling();
        pf.setBorder(new TitledBorder("Polygon Filling"));

        GridLayout gl = new GridLayout(2, 1, 0, 0);
        frame.setLayout(gl);
```

```
        frame.add(pd);
        frame.add(pf);
        frame.setSize(300, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}

class PolygonDrawing extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        Polygon p = new Polygon( );
        p.addPoint(40, 20);
        p.addPoint(70, 40);
        p.addPoint(60, 80);
        p.addPoint(45, 45);
        p.addPoint(20, 60);

        g.drawPolygon(p);

    }
}

class PolygonFilling extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        Polygon p = new Polygon();

        p.addPoint(40,20);
        p.addPoint(70,40);
        p.addPoint(60,80);
        p.addPoint(45,45);
        p.addPoint(20,60);

        g.fillPolygon(p);
    }
}
```

Centering a String using 'FontMetrics' Class:

- public int getAscent( )
- public int getDescent( )
- public int getLeading( )
- public int getHeight( )
- public int stringWidth(String str)

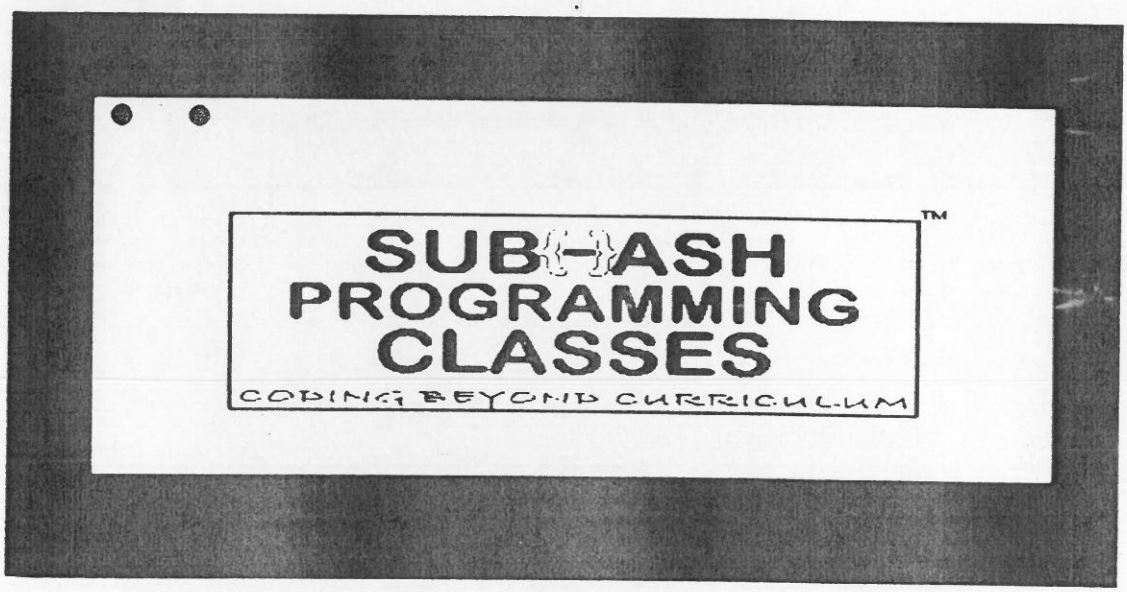





```
import java.awt.*;
import javax.swing.*;

public class TestFontMetrics
{
    public static void main( String [] args )
    {
        JFrame frame = new JFrame( "Testing Font Metrics" );
```

```
        FontMetricsPanel Fp = new FontMetricsPanel();
        Fp.setBackground(Color.WHITE);
        Fp.setFont(new Font( "American Typewriter", Font.BOLD, 30));
        frame.add(Fp);
        frame.setSize(200, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

    }
}

class FontMetricsPanel extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);
        FontMetrics fm = g.getFontMetrics();

        int stringAscent = fm.getAscent();
        int stringWidth = fm.stringWidth("Subhash Programming Classes");

        int x = getWidth()/2 - stringWidth/2;
        int y = getHeight()/2 - stringAscent/2;

        g.drawString( "Subhash Programming Classes", x , y );


    }
}
```

Displaying Images:



```
import java.awt.*;
import javax.swing.*;

public class TestingImageIconImage
{
    public static void main(String [] args )
    {
        DisplayImage di = new DisplayImage();
        JFrame frame = new JFrame();

        frame.add(di);
        frame.setVisible(true);
        frame.setSize(500, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }
}

class DisplayImage extends JPanel
{
    protected void paintComponent( Graphics g )
    {
        super.paintComponent(g);

        ImageIcon ii = new ImageIcon("spc.png");
        Image icon = ii.getImage( );

        g.drawImage(icon, 0, 0, getWidth(), getHeight(), this);
    }
}
```

Handling Exceptions

What is an exception ?

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.

For example:

```
import java.util.*;
public class ExceptionHandling1
{
    public static void main( String [] args )
    {
        System.out.println( "Enter 2 numbers to divide\n" );
        Scanner in = new Scanner(System.in);
        int num1 = in.nextInt();
        int num2 = in.nextInt();

        int res = num1 / num2;

        System.out.println("Result = " + res );
    }
}
```

Output:

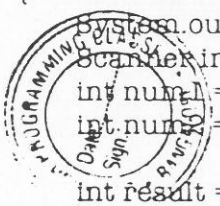
```
$ javac ExceptionHandling1.java
$ java ExceptionHandling1
Enter 2 numbers to divide
11
```

```
2
Result = 5

$ java ExceptionHandling1
Enter 2 numbers to divide
1
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionHandling1.main(ExceptionHandling1.java:22)
```

How do you take control of it ?

Method-1: Un-Professional Way

```
import java.util.*;
public class ExceptionHandling1
{

    public static int Divide( int number1, int number2 )
    {
        if( number2 == 0 )
        {
            System.out.println("Cannot divide by zero");
            System.exit(1);
        }
        return number1 / number2;
    }
    public static void main( String [] args )
    {
        System.out.println( "Enter 2 numbers to divide\n" );
        Scanner in = new Scanner(System.in);
        int num1 = in.nextInt();
        int num2 = in.nextInt();

        int result = Divide(num1, num2);

        System.out.println("Result = " + result );
    }
}
```



Output

```
$ java ExceptionHandling1
Enter 2 numbers to divide

1
0
Cannot divide by zero
$
```

Method-2: Professional Way


```
import java.util.*;
public class ExceptionHandling1
{
    public static int Divide( int number1, int number2 )
    {
        if( number2 == 0 )
        {
            throw new ArithmeticException("Divisor cannot be zero" );
        }
        return number1 / number2;
    }
    public static void main( String [] args )
    {
        System.out.println( "Enter 2 numbers to divide\n" );
        Scanner in = new Scanner(System.in);
        int num1 = in.nextInt();
        int num2 = in.nextInt();

        try
        {
            int result = Divide(num1, num2);

            System.out.println("Result = " + result );
        }
        catch ( ArithmeticException e )
        {
          System.out.println( "Exception Occurred: Cannot Divide by Zero" );
        }

        System.out.println( "I will continue from here" );
    }
}
```

Output:


```
$ java ExceptionHandling1
Enter 2 numbers to divide
1
0
Exception Occurred: Cannot Divide by Zero
I will continue from here
$
```

- The key benefit of exception handling is separating the **exception detection** and the **exception handling**. Detection is done in the called method & handling is done in the calling method.

Let us take another example !

```
import java.util.*;

public class InputMismatchExceptionDemo
{
    public static void main( String [] args )
    {
        Scanner in = new Scanner( System.in );
        boolean continueInput = true;

        do
        {
            try
            {
                System.out.println( "Enter an integer" );
                int number = in.nextInt();

                System.out.println(" The number entered is " + number );

                continueInput = false;
            }
            catch( InputMismatchException ex )
            {
                System.out.println( "Try again.(Incorrect input: an integer
is required)");
                in.nextLine();
            }
        }while(continueInput);
    }
}
```

```
$ java InputMismatchExceptionDemo
Enter an integer
7.8
Try again.(Incorrect input: an integer is required)
Enter an integer
5.6
Try again.(Incorrect input: an integer is required)
Enter an integer
8
The number entered is 8
$
```

Types of Exceptions !

- **Checked Exceptions**
  - Compiler forces the programmer to check and deal with them in a **try-catch** block or declare it in the method header.

- **Unchecked Exceptions**
  - To avoid cumbersome overuse of try-catch blocks, Java does not mandate that you write code to catch or declare unchecked exceptions. **RunTime-Exception**, **Error** and their subclasses are known as unchecked exceptions
  - reflect logical programming errors that are not recoverable. Examples: **NullPointerException**, **ArithmeticException**, **IndexOutOfBoundsException** etc.

Example for 'Checked Exception' :


```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;

public class CheckedExceptionDemo
{
    public static void main(String[] args)
    {

        String thisLine = null;
        try
        {
             BufferedReader br = new BufferedReader(new InputStreamRead-
er(System.in));
            while ((thisLine = br.readLine()) != null)
            {
                System.out.println(thisLine);
            }
        }
        catch(Exception e)
        {
            System.out.println("Cannot Open File");
        }
    }
}
```

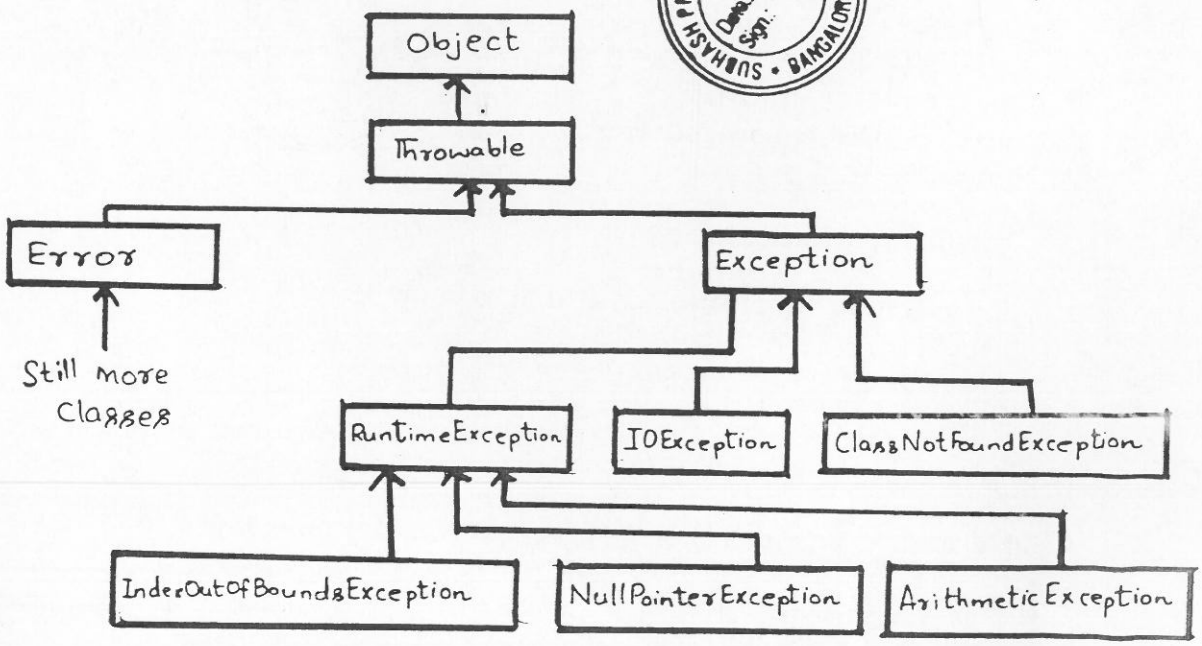
What happens if try-catch is removed from above example:

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.InputStreamReader;

public class CheckedExceptionDemo
{
    public static void main(String[] args)
    {
        String thisLine = null;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        while ((thisLine = br.readLine()) != null)
        {
            System.out.println(thisLine);
        }
    }
}
```



```
$ javac CheckedExceptionDemo.java
CheckedExceptionDemo.java:14: error: unreported exception IOException; must be caught or declared to be thrown
        while ((thisLine = br.readLine()) != null)
                           ^
1 error
```

From the Root of Exception !



Declaring, Throwing, Catching Exceptions:

- If you are throwing, either catch it or declare it.

```
class MyException extends Exception
{
}
class YourException extends Exception
{
}

class Elephant
{
    public void IEatMore( )throws MyException
    {
        ITalkMore();
    }
    public void ITalkMore( ) throws MyException
    {
        IWalkMore( );
    }
    public void IWalkMore( ) throws MyException
    {
        try
        {
            int i = 1;
            if( i == 1 )
            {
                throw new MyException( );
            }
            else
            {
                throw new YourException( );
            }
        }
        catch( YourException e )
        {
            System.out.println("Caught here in I W M");
        }
    }
}

public class ExceptionHandling22
{
    public static void main( String [] args ) throws MyException
    {
        Elephant e = new Elephant();
```

```
        try
        {
            e.IEatMore( );
        }
        catch ( MyException ex )
        {
            System.out.println(" Caught here in Main " );
        }
    }
}
```

Getting Information from Exceptions:

```
class MyException extends Exception
{
    public MyException ( String message )
    {
        super(message);
    }
}
class YourException extends Exception
{
}

class Elephant
{
    public void IEatMore( ) throws MyException
    {
        ITalkMore();
    }
    public void ITalkMore( ) throws MyException
    {
        IWalkMore( );
    }
    public void IWalkMore( ) throws MyException
    {
        try
        {
            int i = 1;
            if( i == 1 )
            {
                throw new MyException("Learning Exceptions");
            }
            else
            {
                throw new YourException();
            }
        }
```

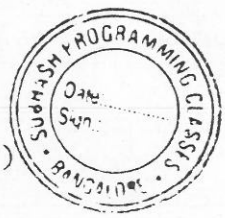
```
        catch( YourException e )
        {
            System.out.println("Caught here in I W M");
        }
    }
}

public class ExceptionHandling22
{
    public static void main( String [] args ) throws MyException
    {
        Elephant e = new Elephant();
        try
        {
            e.IEatMore();
        }
        catch( MyException ex )
        {
            System.out.println( ex.getMessage());
            System.out.println( ex.toString());
            ex.printStackTrace();
        }
    }
}
```

Output:

```
$ java ExceptionHandling22.java
Learning Exceptions
MyException: Learning Exceptions
MyException: Learning Exceptions
        at Elephant.IWalkMore(ExceptionHandling22.java:40)
        at Elephant.ITalkMore(ExceptionHandling22.java:31)
        at Elephant.IEatMore(ExceptionHandling22.java:27)
        at ExceptionHandling22.main(ExceptionHandling22.java:62)
```

Multiple catch blocks:

```
import java.util.*;

public class ExceptionHandling3
{
    public static void main( String [] args )
    {
        try
        {
            if( args.length == 0 )
```



```
            {
                int a = 5 / 0;
            }
            else
            {
                int c[] = new int [5];
                c[25] = 6;
            }
        }
        catch( ArithmeticException e )
        {
            System.out.println( "Divide by Zero" );
            System.out.println( e );
        }
        catch( ArrayIndexOutOfBoundsException e )
        {
            System.out.println( "Out of Bounds" );
            System.out.println( e );
        }
        System.out.println( "I am outside try" );
    }
}
```

Output:


```
$ java ExceptionHandling3
Divide by Zero
java.lang.ArithmeticException: / by zero
I am outside try
$ java ExceptionHandling3 34
Out of Bounds
java.lang.ArrayIndexOutOfBoundsException: 25
I am outside try
```

Beware !!!


```
class MyException extends Exception
{
}
class MyMyException extends MyException
{
}
class MyMyMyException extends MyException
{
}

class Elephant
{
    public void IEatMore( ) throws MyException
```

```
    {
        throw new MyMyException();
    }
}

public class ExceptionHandling44
{
    public static void main( String [] args )
    {
        Elephant e = new Elephant();
        try
        {
            e.IEatMore();
        }
        catch( MyException ex )
        {
            System.out.println( "MyException: " + ex.getMessage());
        }
        catch( MyMyException ex )
        {
            System.out.println( "MyMyException: " + ex.getMessage());
        }
    }
}
```


```
$ javac ExceptionHandling44.java
ExceptionHandling44.java:44: error: exception MyMyException has already
been caught
            catch( MyMyException ex )
            ^
1 error
$
```

Re-throwing Exceptions:

```
import java.util.*;
import java.io.*;

public class ExceptionHandling5
{
    public static void Call( ) throws IOException
    {
        try
        {
            //Some error code
            throw new IOException( "Checking" );
        }
        catch( IOException e )
```

```
        {
            throw e; //re-throwing exception
        }
    }
    public static void main( String [] args )
    {
        try
        {
            Call();
        }
        catch( IOException e )
        {
            System.out.println( "Recaught: " + e );
        }
    }
}
```

```
$ java ExceptionHandling5
Recaught: java.io.IOException: Checking
$
```

The 'finally' Clause:

- The **'finally'** clause is always executed regardless whether an exception occurred or not.

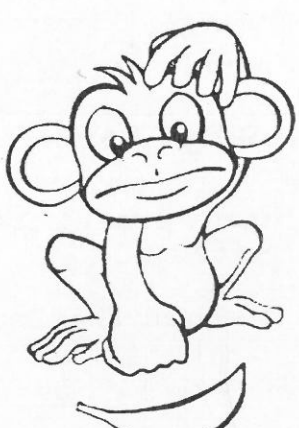
```
public class FinallyBlockDemo
{
    public static void main(String[] args)
    {
        try
        {
            int i = 10/0;
        }
        catch(Exception ex)
        {
            System.out.println("Inside 1st catch Block");
        }
        finally
        {
            System.out.println("Inside 1st finally block");
        }

        try
        {
            int i = 10/10;
        }
        catch(Exception ex)
```

```
        {
            System.out.println("Inside 2nd catch Block");
        }
        finally
        {
            System.out.println("Inside 2nd finally block");
        }
    }
}
```

Output:

```
$ java FinallyBlockDemo
Inside 1st catch Block
Inside 1st finally block
Inside 2nd finally block
$
```

Guess the Output:



1.

```
public class EHTestOne
{
    public static void main( String [] args )
    {
        for( int i = 0; i < 5; i++ )
        {
            System.out.print( i + " " );
            try
            {
                System.out.println( 1 / i );
            }
            catch( Exception ex )
            {
            }
        }
    }
}
```

2.

```
public class EHTestTwo
{
    public static void main( String [] args )
    {
        try
        {
            for( int i = 0; i < 5; i++ )
```

```
                {
                    System.out.print( i + " " );
                    System.out.println( 1 / i );
                }
            }
            catch( Exception ex )
            {

            }
        }
}
```

3.

```
public class EHTestThree
{
    public static void main( String [] args )
    {
        int [] list = new int [5];
        System.out.println(list[-1]);
    }
}
```

4.

```
public class EHTestFour
{
    public static void main( String [] args )
    {
        String s = "abc";
        System.out.println(s.charAt(-2));
    }
}
```

5.

```
public class EHTestFive
{
    public static void main( String [] args )
    {
        int i = 10;
        System.out.println(i.toString());
    }
}
```

6.

```
public class EHTestSix
{
```

```
    public static void main( String [] args )
    {
        float f = 4.5f;
        System.out.println( f / 0 );
    }
}
```

7.

```
public class EHTestOne
{
    public static void main( String [] args )
    {
        try
        {
            int [] list = new int[5];
            System.out.println( list[-1]);
        }
        catch( ArithmeticException ex )
        {
            System.out.println( "ArithmeticException" );
        }
        catch( RuntimeException ex )
        {
            System.out.println( "RunTimeException" );
        }
        catch( Exception ex )
        {
            System.out.println( "Exception" );
        }
    }
}
```

8.

```
public class EHTestOne
{
    public static void main( String [] args )
    {
        try
        {
            System.out.println( "Before the method call");
            method();
            System.out.println("After the method call");
        }
        catch( ArithmeticException ex )
        {
            System.out.println( "ArithmeticException" );
        }
        catch( RuntimeException ex )
        {
```

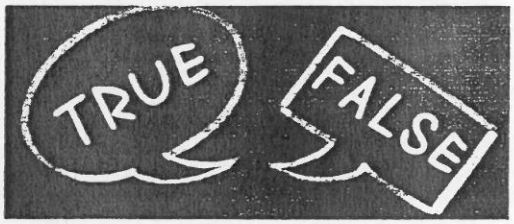
```
            System.out.println( "RunTimeException" );
        }
        catch( Exception ex )
        {
            System.out:println( "Exception" );
        }
        System.out.println("End of main");
    }

    public static void method( ) throws Exception
    {
        System.out.println(0/0);
    }
}
```

9.

```
public class EHTestOne
{
    public static void main( String [] args )
    {
        try
        {
            System.out.println( "Before the method call");
            method();
            System.out.println("After the method call");
        }
        catch( RuntimeException ex )
        {
            System.out.println( "RunTimeException in main" );
        }
        catch( Exception ex )
        {
            System.out.println( "Exception in main" );
        }
    }

    public static void method( ) throws Exception
    {
        try
        {
            int i = 1/0;
            System.out.println( "i is " + i );
        }
        catch( RuntimeException ex )
        {
            System.out.println( "RuntimeException in method()");
            throw ex;
        }
```

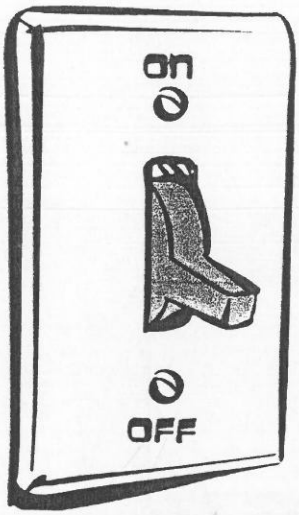
```
        catch( Exception ex )
        {
            System.out.println( "Exception in method()" );
            throw ex;
        }
    }
}
```

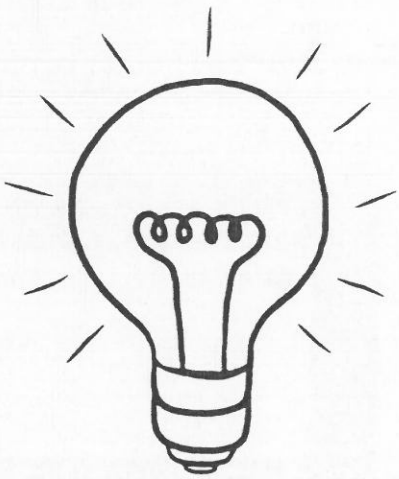
State True or False :



1. When throwing exceptions, either handle it or declare that it is thrown :
2. All exceptions are checked exceptions :
3. '**finally**' block is executed only when **catch** block is entered :

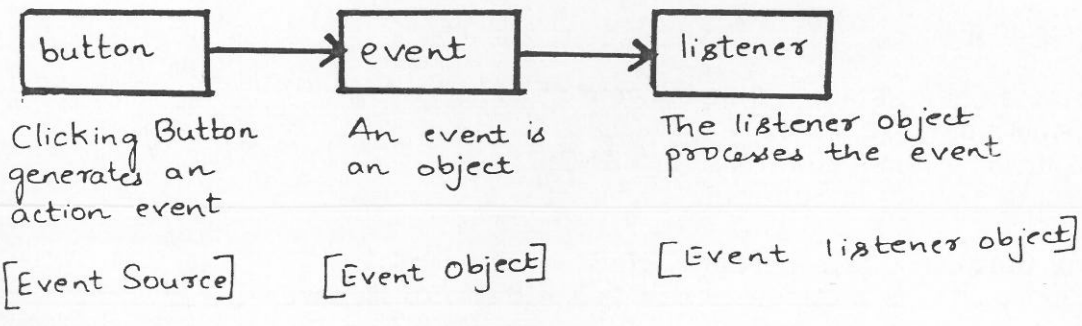
Handling Events





Steps needed for Event Handling:

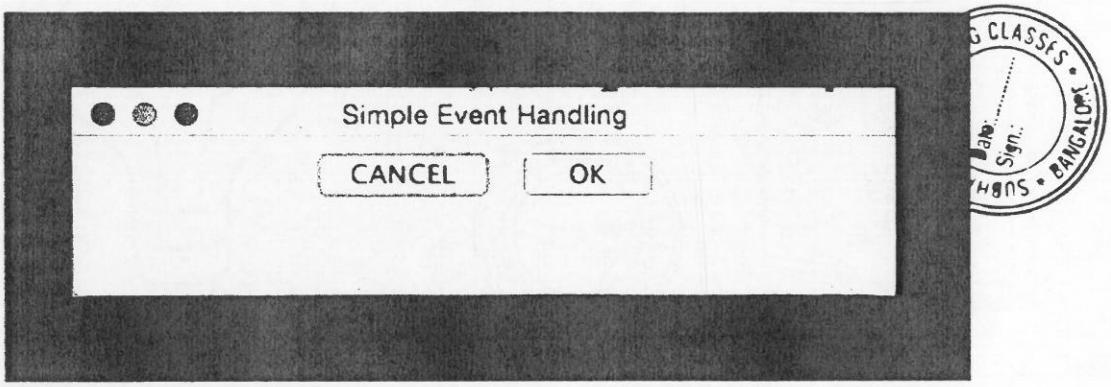
- Create your graphics - Frames, Panels, Buttons, Text Fields or whatever. They are your **event** sources.
- Create **listener** classes. They are the ones who listen to your **event**s.
  - listener classes should implement respective listener interfaces
  - Make your listener classes as inner classes
- Register the **listener**s with your **event** sources.
- Generate the **event**.
- Enjoy by seeing your **listener**s perform.



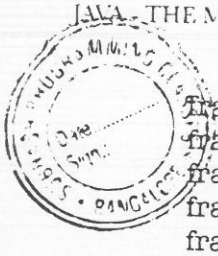
A simple Event Handling Example:

Facts to be known:

User Action	: CLICK A BUTTON
Event Source	: JButton
Event Type	: ActionEvent
Listener Interface	: ActionListener
Listener Interface Methods	: actionPerformed(ActionEvent e)



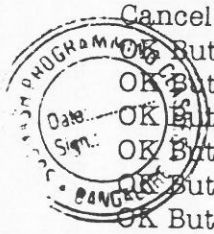

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class SimpleEventHandling
{
    JFrame frame;
    JButton button;
    JPanel panel;

    public static void main( String [] args )
    {
        SimpleEventHandling seh = new SimpleEventHandling( );
        seh.go( );
    }

    public void go( )
    {
        frame = new JFrame( "Simple Event Handling" );
        panel = new JPanel();
        JButton cancel = new JButton("CANCEL");
        JButton ok   = new JButton("OK" );

        ok.addActionListener(new OkButtonListener() );
        cancel.addActionListener(new CancelButtonListener() );
        panel.add(cancel);
        panel.add(ok);
```



```
        frame.add(panel);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,100);
        frame.setLocationRelativeTo(null);
    }

    class OkButtonListener implements ActionListener
    {
        public void actionPerformed( ActionEvent e )
        {
            System.out.println( "OK Button Clicked" );
        }
    }

    class CancelButtonListener implements ActionListener
    {
        public void actionPerformed( ActionEvent e )
        {
            System.out.println( "Cancel Button Clicked" );
        }
    }
}
```

Ouput:

```
$ java SimpleEventHandling
OK Button Clicked
Cancel Button Clicked
OK Button Clicked
OK Button Clicked
OK Button Clicked
OK Button Clicked
OK Button Clicked
OK Button Clicked
OK Button Clicked
Cancel Button Clicked
Cancel Button Clicked
Cancel Button Clicked
Cancel Button Clicked

$
```

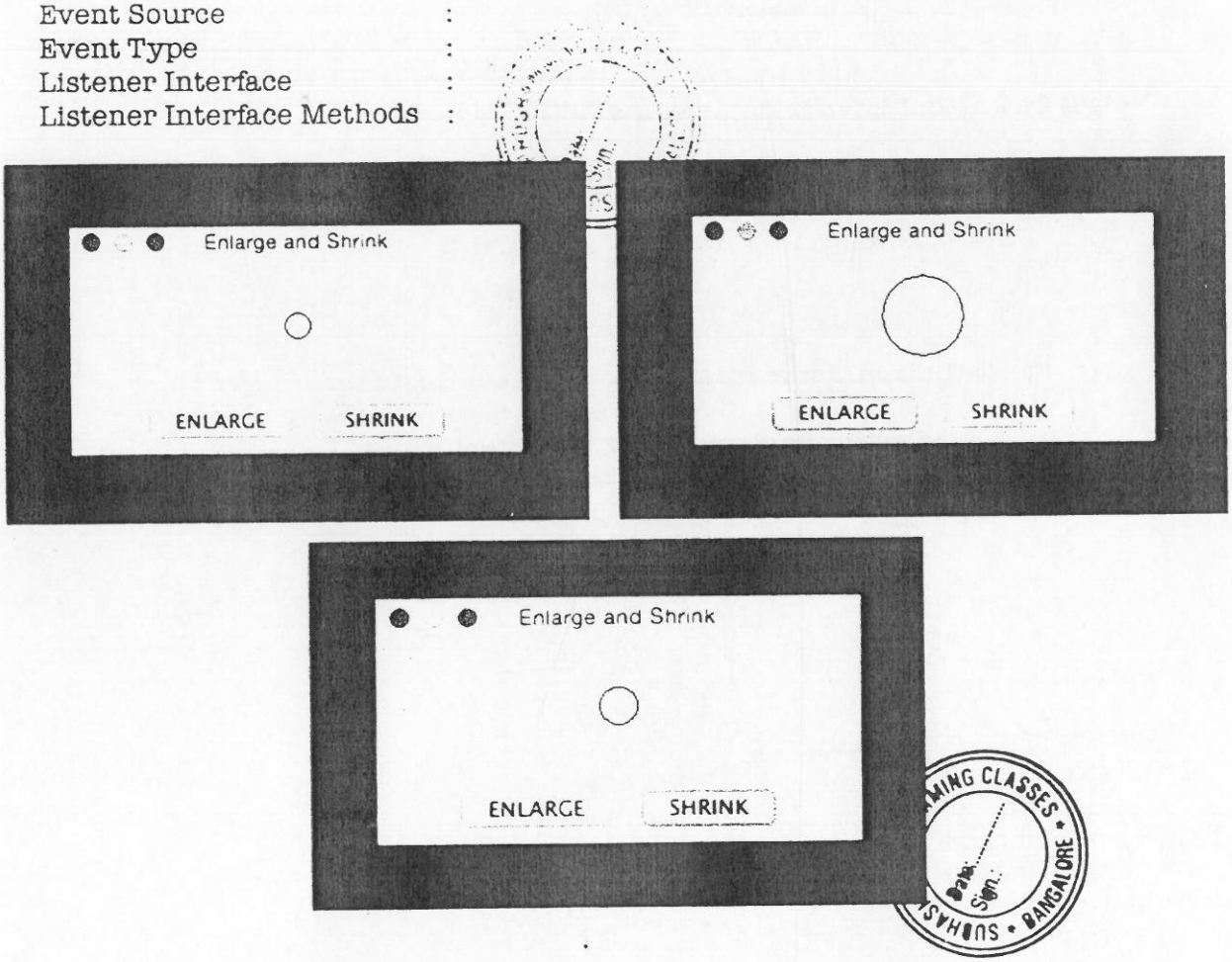


Another Event Handling Example:

Fill in the blanks below:

Facts to be known:

User Action :
Event Source :
Event Type :
Listener Interface :
Listener Interface Methods :




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

public class EnlargeAndShrink
{
    JFrame frame;
    CirclePanel circlePanel;

    public static void main( String [] args )
    {
        EnlargeAndShrink es = new EnlargeAndShrink();
        es.go( );
    }
    public void go( )
```

```
    {
        frame = new JFrame("Enlarge and Shrink" );
        JPanel buttonPanel = new JPanel();
        circlePanel = new CirclePanel();
        JButton enlarge = new JButton( "ENLARGE");
        JButton shrink  = new JButton( "SHRINK");
        buttonPanel.add(enlarge);
        buttonPanel.add(shrink);
        enlarge.addActionListener(new EnlargeListener());
        shrink.addActionListener(new ShrinkListener());
        frame.add(buttonPanel, BorderLayout.SOUTH);
        frame.add(circlePanel, BorderLayout.CENTER);
        frame.setVisible(true);
        frame.setSize(500, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

    }

    class EnlargeListener implements ActionListener
    {
        public void actionPerformed( ActionEvent e )
        {
            circlePanel.enlarge();
        }
    }
    class ShrinkListener implements ActionListener
    {
        public void actionPerformed( ActionEvent e )

        {
            circlePanel.shrink();
        }
    }


}

class CirclePanel extends JPanel
{
    private int radius = 5;

    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        int x = getWidth( )/2 - radius;
        int y = getHeight( )/2 - radius;
```

```
        g.drawOval( x, y, 2 * radius, 2 *radius);
    }

    public void enlarge( )
    {
        radius++;
        repaint( );
    }
    public void shrink( )
    {
        radius--;
        repaint( );
    }
}
```

Anonymous Class Listeners:

- An anonymous inner class is an inner class without a name. It combines defining an inner class and creating an instance of the class into one step.

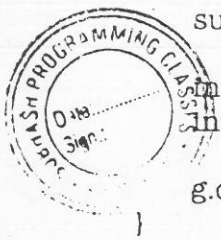
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EnlargeAndShrink
{
    JFrame frame;
    CirclePanel circlePanel;

    public static void main( String [] args )
    {
        EnlargeAndShrink es = new EnlargeAndShrink();
        es.go( );
    }
    public void go( )
    {
        frame = new JFrame("Enlarge and Shrink" );
        JPanel buttonPanel = new JPanel();
        circlePanel = new CirclePanel();
        JButton enlarge = new JButton( "ENLARGE");
        JButton shrink  = new JButton( "SHRINK");
        buttonPanel.add(enlarge);
        buttonPanel.add(shrink);
        enlarge.addActionListener(new ActionListener( ) {
                        public void actionPerformed (ActionEvent e)
                        {
                            circlePanel.enlarge();
                        }
```

```
                    } );
            shrink.addActionListener(new ActionListener ( ) {
                        public void actionPerformed ( ActionEvent e )
                        {
                            circlePanel.shrink();
                        }
                    });
            frame.add(buttonPanel, BorderLayout.SOUTH);
            frame.add(circlePanel, BorderLayout.CENTER);
            frame.setVisible(true);
            frame.setSize(500, 200);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLocationRelativeTo(null);

    }
}
class CirclePanel extends JPanel
{
    private int radius = 5;


    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        int x = getWidth()/2 - radius;
        int y = getHeight()/2 - radius;

        g.drawOval( x, y, 2 * radius, 2 *radius);
    }

    public void enlarge( )
    {
        radius++;
        repaint( );
    }
    public void shrink( )
    {
        radius--;
        repaint( );
    }
}
```

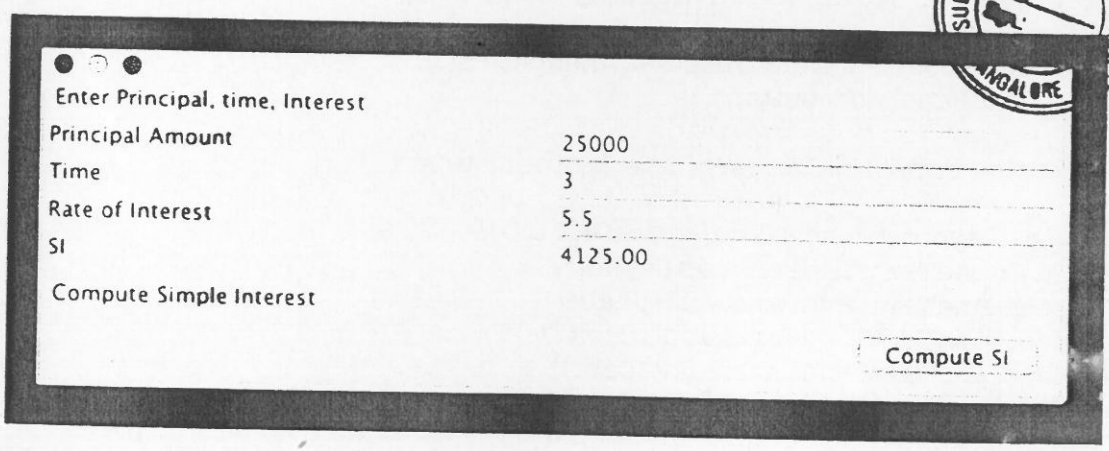
Multiple Events - Single Listener:

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

public class EnlargeAndShrink
{
    JFrame frame;
    CirclePanel circlePanel;
    JButton enlarge;
    JButton shrink;

    public static void main( String [] args )
    {
        EnlargeAndShrink es = new EnlargeAndShrink();
        es.go( );
    }
    public void go( )
    {
        frame = new JFrame("Enlarge and Shrink" );
        JPanel buttonPanel = new JPanel();
        circlePanel = new CirclePanel();
        enlarge = new JButton( "ENLARGE");
        shrink  = new JButton( "SHRINK");
        buttonPanel.add(enlarge);
        buttonPanel.add(shrink);
        enlarge.addActionListener(new EnlargeAndShrinkListener());
        shrink.addActionListener(new EnlargeAndShrinkListener());
        frame.add(buttonPanel, BorderLayout.SOUTH);
        frame.add(circlePanel, BorderLayout.CENTER);
        frame.setVisible(true);
        frame.setSize(500, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

    }

    class EnlargeAndShrinkListener implements ActionListener
    {
        public void actionPerformed(ActionEvent ae )
        {
            if( ae.getSource( ) == enlarge )
                circlePanel.enlarge();
            if( ae.getSource( ) == shrink )
                circlePanel.shrink();
        }
    }
}
```

```
class CirclePanel extends JPanel
{
    private int radius = 5;

    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);

        int x = getWidth()/2 - radius;
        int y = getHeight()/2 - radius;

        g.drawOval( x, y, 2 * radius, 2 *radius);
    }

    public void enlarge( )
    {
        radius++;
        repaint( );
    }
    public void shrink( )
    {
        radius--;
        repaint( );
    }
}
```



Simple Interest Calculator:



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class SimpleInterestEventHandling
{
    JLabel pLabel = new JLabel("Principal Amount" );
    JTextField pTextField = new JTextField();
    JLabel tLabel = new JLabel( "Time" );
    JTextField tTextField = new JTextField( );
    JLabel rateLabel = new JLabel ("Rate of Interest");
    JTextField rTextField = new JTextField( );
    JLabel siLabel = new JLabel( "SI" );
    JTextField siTextField = new JTextField();

    public static void main( String [] args )
    {
        SimpleInterestEventHandling si = new SimpleInterestEventHandling( );
        si.go( );
    }
    public void go( )
    {
        JFrame frame = new JFrame( );

        JPanel labelAndTextFieldPanel = new JPanel( );
        JPanel buttonPanel = new JPanel( new FlowLayout(FlowLayout.RIGHT ));

        labelAndTextFieldPanel.setBorder( new TitledBorder( "Enter Principal, time,
Interest" ));
        buttonPanel.setBorder( new TitledBorder( "Compute Simple Interest" ));
        frame.add(labelAndTextFieldPanel, BorderLayout.CENTER);
        frame.add(buttonPanel, BorderLayout.SOUTH);

        JButton button = new JButton( "Compute SI");
        buttonPanel.add(button);

        button.addActionListener( new ButtonListener());

        GridLayout gl = new GridLayout(4,2,0,0);
        labelAndTextFieldPanel.setLayout(gl);
        labelAndTextFieldPanel.add(pLabel);
        labelAndTextFieldPanel.add(pTextField);
        labelAndTextFieldPanel.add(tLabel);
        labelAndTextFieldPanel.add(tTextField);
        labelAndTextFieldPanel.add(rateLabel);
        labelAndTextFieldPanel.add(rTextField);
        labelAndTextFieldPanel.add(siLabel);
        labelAndTextFieldPanel.add(siTextField);
```

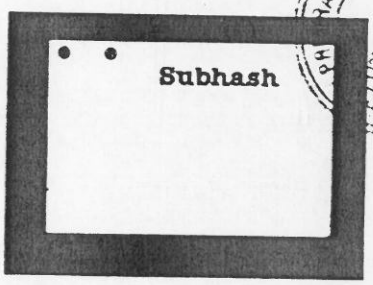
```
        frame.setSize(600, 200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
    }

    class ButtonListener implements ActionListener
    {
        public void actionPerformed( ActionEvent ae )
        {
            double pa = Double.parseDouble(pTextField.getText());
            int time  = Integer.parseInt(tTextField.getText());
            double roi = Double.parseDouble(rTextField.getText());

            double si = (pa * time * roi)/100;

            siTextField.setText(String.format("%.2f", si ));
        }
    }
}
```

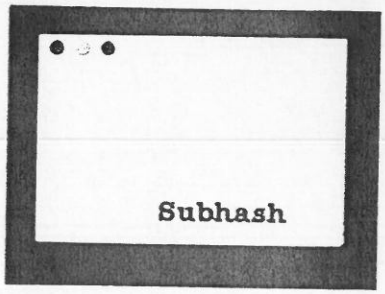
Mouse Events & Listeners:

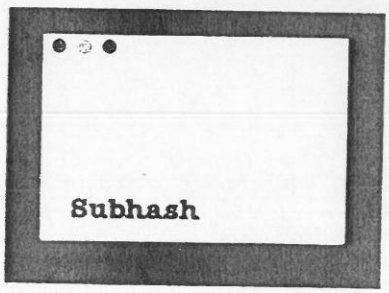

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class MouseEventHandling
{
    public static void main( String [] args )
    {
        MouseEventHandling me = new MouseEventHandling( );
        me.go( );
    }
    public void go( )
    {
        JFrame frame = new JFrame( );

        MouseEventPanel panel = new MouseEventPanel();
        frame.add(panel);
        panel.addMouseMotionListener(panel);
```

```
        frame.setSize(300,200);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);


    class MouseEventPanel extends JPanel implements MouseMotionListener
    {
        private int x = 5;
        private int y = 10;

        public void mouseDragged( MouseEvent me )
        {
            x = me.getX( );
            y = me.getY( );
            repaint( );
        }
        protected void paintComponent( Graphics g )
        {
            super.paintComponent(g);
            g.drawString( "Welcome", x, y );
        }

        public void mouseMoved( MouseEvent me )
        {
        }
    }
}
```

Key Events & Listeners:





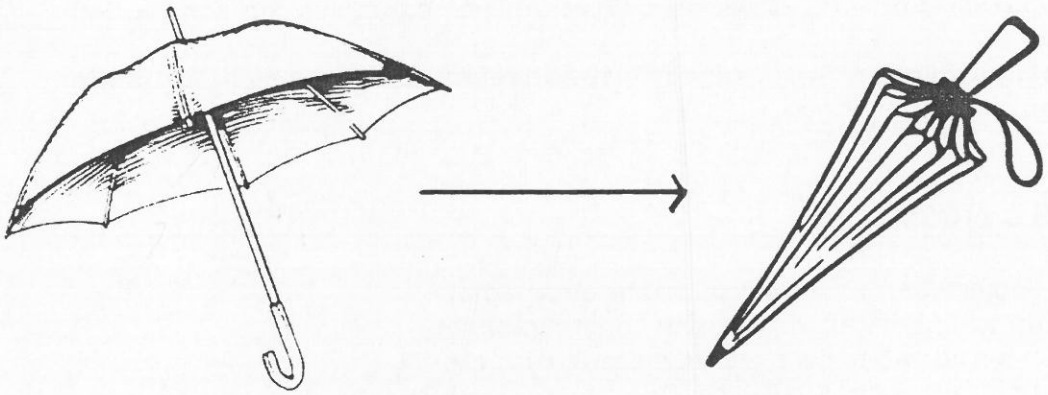


```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyEventHandling
{
    public static void main( String [] args )
    {
        KeyEventHandling ke = new KeyEventHandling( );
        ke.go( );
    }
    public void go( )
    {
        JFrame frame = new JFrame( );

        KeyEventPanel  panel = new KeyEventPanel();
        panel.addKeyListener(panel);
        frame.add(panel);

        panel.setFocusable(true);
        frame.setVisible(true);
        frame.setLocationRelativeTo(null);
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    class KeyEventPanel extends JPanel implements KeyListener
    {
        private int x = 5;
        private int y = 5;
        private char keyChar = 'A';

        protected void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            g.setFont(new Font("American Typewriter", Font.BOLD, 24));
            g.drawString("Subhash", x, y );
        }
        public void keyPressed( KeyEvent e )
        {
            switch( e.getKeyCode() )
            {
                case KeyEvent.VK_DOWN: y += 10;

                                break;
                case KeyEvent.VK_UP:  y -= 10;
                                break;
                case KeyEvent.VK_LEFT: x -= 10;
                                break;
```

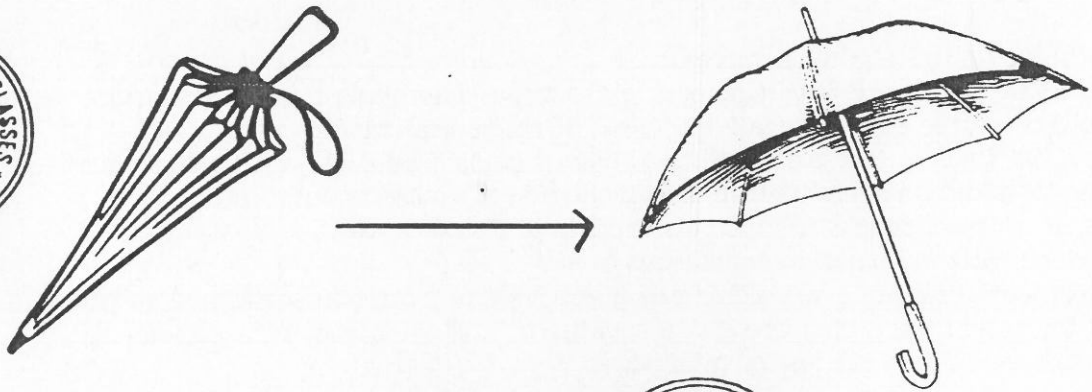
```
                case KeyEvent.VK_RIGHT: x += 10;
                                        break;
                default:        e.getKeyChar();
            }
            repaint();
        }
        public void keyReleased( KeyEvent e) { }
        public void keyTyped( KeyEvent e ) { }
    }

}
```

Serialization, De-Serialization & Text I/O

What is Serialization ?



What is De-Serialization ?



Two ways of saving objects :

1. **Serialization:**
   - If saved objects are used by the program that generated it.
2. **Saving to a file:**
   - If saved objects are used by the other programs.

- **Serialization** referes to saving the state (instance variable values) of an object in the heap to a local file in binary format.
- **De-Serialization** refers to getting back the object on the heap from its serialised state.
- To serialize an object, its class should implement '**Serializable**' interface.
- '**Serializable**' is a marker interface.
- If any superclass of a class is '**Serializable**' its subclass is automatically '**Serializable**' even if it does not implement '**Serializable**' .
- If an instance variable is a reference to another object, then, make sure even the reference variable class should implement '**Serializable**'.
- If you do not wish to make an object's state '**Serializable**' make it **transient**.
- **'FileInputStream'** and **'FileOutputStream'** are examples for **connection streams**.
- **'ObjectInputStream'** and **'ObjectOutputStream'** are examples for **chain streams**.

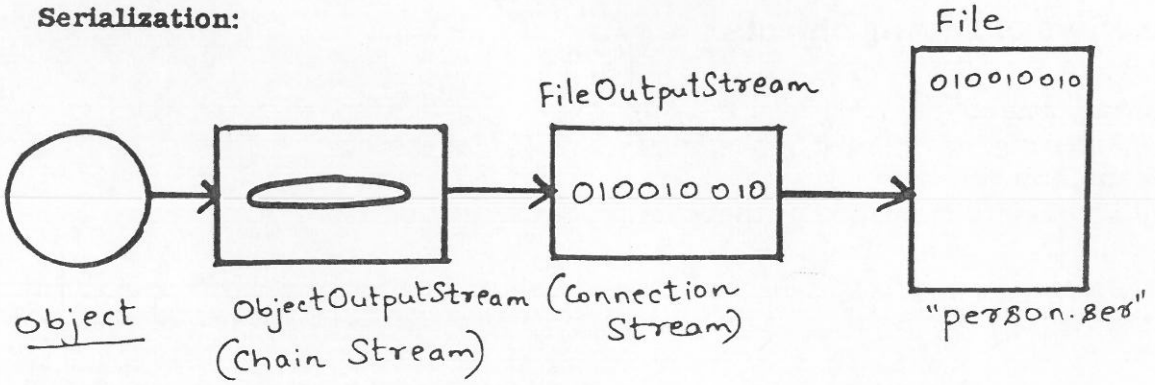
SPECIAL NOTE:

- If the object has a non-serializable class somewhere up its inheritance tree, the constructor for that non-serializable class will run along with other constructors above that.

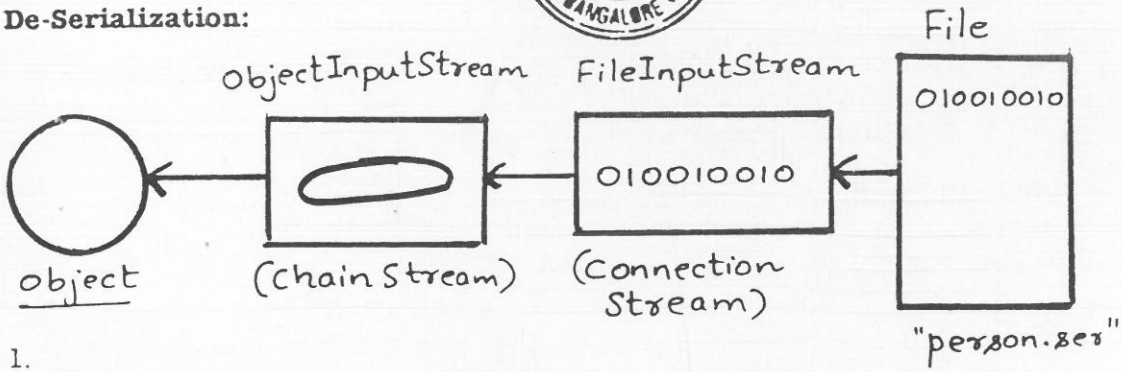
What happens during De-Serialization :

1. The object is read from the stream.
2. The JVM determine's the objects class type. The class type information would be stored in the serialized object during serialization.
3. The JVM tries to find and load the object's class. The JVM would throw an exception and de-serialization fails if the object's class is not found.
4. The new object is given space on the heap. Please note the constructor of the deserialized object does not run.
5. The object's instance variables are given values from the serialized state. The transient variables are given a value of null for object references and respective default values for primitives.

Serialization:



De-Serialization:



1.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;

    Person( String n, int i )
    {
        name = n;
        id = i;
    }
    String getName( )
    {
        return name;
    }
    int getId( )
    {
        return id;
    }
}

public class SerializationTest_1
{

    public static void main( String [] args )
    {
        Person subhash = new Person("Subhash", 18);
        Person sumesh  = new Person("Sumesh", 19);
        Person rajesh  = new Person("Rajesh", 20);

        try
        {
            FileOutputStream foutput = new FileOutputStream("person.ser");
            ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

            ooutput.writeObject(subhash);
            ooutput.writeObject(sumesh);
            ooutput.writeObject(rajesh);
```

```
            ooutput.close();
        }
        catch( IOException ex )
        {
            ex.printStackTrace();
        }
        subhash = null;
        sumesh  = null;
        rajesh  = null;

        try
        {
            FileInputStream finput = new FileInputStream("person.ser");
            ObjectInputStream oinput = new ObjectInputStream(finput);
            Object one   = oinput.readObject();
            Object two   = oinput.readObject();
            Object three = oinput.readObject();
            oinput.close();

            subhash = (Person)one;
            sumesh  = (Person)two;
            rajesh  = (Person)three;
        }
        catch( Exception ex )
        {
            ex.printStackTrace();
        }
        System.out.println( subhash.getName());
        System.out.println( sumesh.getName());
        System.out.println( rajesh.getName());
    }
}
```

Output:

```
$ java SerializationTest_1
Subhash
Sumesh
Rajesh
$
```

2.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;
    Eyes e = new Eyes();
```

```
    Person( String n, int i )
    {
        name = n;
        id = i;
    }
    String getName( )
    {
        return name;
    }
    int getId( )
    {
        return id;
    }
    Eyes getEye( )
    {
        return e;
    }
}

class Eyes implements Serializable
{
    int size;
    public Eyes( )
    {
        size = 6;
    }
    void changeSize()
    {
        ++size;
    }
    int getSize()
    {
        return size;
    }
}


public class SerializationTest_2
{

    public static void main( String [] args )
    {
        Person subhash = new Person( "Subhash", 18);
        Person sumesh  = new Person("Sumesh", 19);
        Person rajesh  = new Person("Rajesh", 20);
        subhash.e.changeSize();
        sumesh.e.changeSize();
        rajesh.e.changeSize();

        System.out.println( subhash.e.getSize());
        System.out.println( sumesh.e.getSize());
        System.out.println( rajesh.e.getSize());
```

```
        try
        {
            FileOutputStream foutput = new FileOutputStream("person.txt");
            ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

            ooutput.writeObject(subhash);
            ooutput.writeObject(sumesh);
            ooutput.writeObject(rajesh);

            ooutput.close();
        }
        catch( IOException ex )
        {
            ex.printStackTrace();
        }
        subhash = null;
        sumesh  = null;
        rajesh  = null;

        try
        {
            FileInputStream finput = new FileInputStream("person.txt");
            ObjectInputStream oinput = new ObjectInputStream(finput);
            Object one   = oinput.readObject();
            Object two   = oinput.readObject();
            Object three = oinput.readObject();

            oinput.close();

            subhash = (Person)one;
            sumesh  = (Person)two;
            rajesh  = (Person)three;
        }
        catch( Exception ex )
        {
            ex.printStackTrace();
        }


        System.out.println( subhash.getName());
        System.out.println( sumesh.getName());
        System.out.println( rajesh.getName());

        System.out.println( subhash.getId());
        System.out.println( sumesh.getId());
        System.out.println( rajesh.getId());

        System.out.println( subhash.e.getSize());
        System.out.println( sumesh.e.getSize());
        System.out.println( rajesh.e.getSize());
    }
}
```

Output:

```
$ java SerializationTest_2
7
7
7
Subhash
Sumesh
Rajesh
18
19
20
7
7
7
$
```

3.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;
    transient Eyes e = new Eyes( );

    Person( String n, int i )
    {
        name = n;
        id = i;
    }
    String getName( )
    {
        return name;
    }
    int getId( )
    {
        return id;
    }
    Eyes getEye( )
    {
        return e;
    }
}

class Eyes
{
}

public class SerializationTest_3
{

    public static void main( String [] args )
```

```
{
    Person subhash = new Person("Subhash", 18);
    Person sumesh  = new Person("Sumesh", 19);
    Person rajesh  = new Person("Rajesh", 20);

    try
    {
        FileOutputStream foutput = new FileOutputStream("person.txt");
        ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

        ooutput.writeObject(subhash);
        ooutput.writeObject(sumesh);
        ooutput.writeObject(rajesh);

        ooutput.close();
    }
    catch( IOException ex )

    {
        ex.printStackTrace();
    }
    subhash = null;
    sumesh  = null;
    rajesh  = null;

    try
    {
        FileInputStream finput = new FileInputStream("person.txt");
        ObjectInputStream oinput = new ObjectInputStream(finput);
        Object one   = oinput.readObject( );
        Object two   = oinput.readObject( );
        Object three = oinput.readObject( );
        oinput.close();

        subhash = (Person)one;
        sumesh  = (Person)two;
        rajesh  = (Person)three;
    }
    catch( Exception ex )
    {
        ex.printStackTrace();
    }


    System.out.println( subhash.getName( ));
    System.out.println( sumesh.getName( ));
    System.out.println( rajesh.getName( ));

    System.out.println( subhash.getId( ));
    System.out.println( sumesh.getId( ));
    System.out.println( rajesh.getId( ));

    System.out.println( subhash.getEye( ));
```

```
            System.out.println( sumesh.getEye( ));
            System.out.println( rajesh.getEye( ));
        }
}
```

Output:

```
$ java SerializationTest_3
Subhash
Sumesh
Rajesh
18
19
20
null
null
null
$
```

4.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;
    Eyes e = new Eyes();

    Person( String n, int i )
    {
        name = n;
        id = i;
    }
    String getName( )
    {
        return name;
    }
    int getId( )
    {
        return id;
    }
    Eyes getEye( )
    {
        return e;
    }
}

class Eyes
{
}

public class SerializationTest_3
{
```

```
public static void main( String [] args )
{
    Person subhash = new Person("Subhash", 18);
    Person sumesh  = new Person("Sumesh", 19);
    Person rajesh  = new Person("Rajesh", 20);

    try
    {
        FileOutputStream foutput = new FileOutputStream("person.txt");
        ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

        ooutput.writeObject(subhash);
        ooutput.writeObject(sumesh);
        ooutput.writeObject(rajesh);

        ooutput.close();
    }
    catch( IOException ex )

    {
        ex.printStackTrace();
    }
    subhash = null;
    sumesh  = null;
    rajesh  = null;

    try
    {
        FileInputStream finput = new FileInputStream("person.txt");
        ObjectInputStream oinput = new ObjectInputStream(finput);
        Object one   = oinput.readObject( );
        Object two   = oinput.readObject( );
        Object three = oinput.readObject( );
        oinput.close();

        subhash = (Person)one;
        sumesh  = (Person)two;
        rajesh  = (Person)three;
    }
    catch( Exception ex )
    {
        ex.printStackTrace();
    }


    System.out.println( subhash.getName( ));
    System.out.println( sumesh.getName( ));
    System.out.println( rajesh.getName( ));

    System.out.println( subhash.getId( ));
    System.out.println( sumesh.getId( ));
    System.out.println( rajesh.getId( ));
```

```
        System.out.println( subhash.getEye( ));
        System.out.println( sumesh.getEye( ));
        System.out.println( rajesh.getEye( ));
    }
}
```

Output:
$ (Run-Time Exception would be generated)

5.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;
    Eyes e = new Eyes();

    Person( String n, int i )
    {
        System.out.println( "Constructor Called" );
        name = n;
        id = i;
    }

    String getName( )
    {
        return name;
    }

    int getId( )
    {
        return id;
    }
    Eyes getEye( )
    {
        return e;
    }
}

class Eyes implements Serializable
{
    Eyes( )
    {
        System.out.println( "Eyes Constructor Called" );
    }
}

public class SerializationTest_4
{

    public static void main( String [] args )
```

```
{
    Person subhash = new Person("Subhash", 18);
    Person sumesh  = new Person("Sumesh", 19);
    Person rajesh  = new Person("Rajesh", 20);

    try
    {
        FileOutputStream foutput = new FileOutputStream("person.txt");
        ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

        ooutput.writeObject(subhash);
        ooutput.writeObject(sumesh);
        ooutput.writeObject(rajesh);

        ooutput.close();
    }
    catch( IOException ex )
    {
        ex.printStackTrace();
    }
    subhash = null;
    sumesh  = null;
    rajesh  = null;

    try
    {
        FileInputStream finput = new FileInputStream("person.txt");
        ObjectInputStream oinput = new ObjectInputStream(finput);
        Object one   = oinput.readObject( );
        Object two   = oinput.readObject( );
        Object three = oinput.readObject( );

        oinput.close();

        subhash = (Person)one;
        sumesh  = (Person)two;
        rajesh  = (Person)three;
    }
    catch( Exception ex )
    {
        ex.printStackTrace();
    }


    System.out.println( subhash.getName( ));
    System.out.println( sumesh.getName( ));
    System.out.println( rajesh.getName( ));

    System.out.println( subhash.getId( ));
    System.out.println( sumesh.getId( ));
    System.out.println( rajesh.getId( ));

    System.out.println( subhash.getEye( ));
```

```
        System.out.println( sumesh.getEye( ));
        System.out.println( rajesh.getEye( ));
    }
}
```

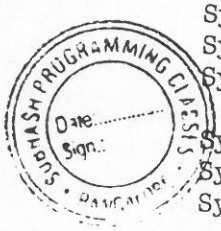
```
$
Eyes Constructor Called
Constructor Called
Eyes Constructor Called
Constructor Called
Eyes Constructor Called
Constructor Called
Subhash
Sumesh
Rajesh
18
19
20
Eyes@135fbaa4
Eyes@45ee12a7
Eyes@330bedb4
$
```

6.

```
import java.io.*;
class Person implements Serializable
{
    String name;
    int id;
    Eyes e = new Eyes();

    Person( String n, int i )
    {
        System.out.println( "Person Constructor Called" );
        name = n;
        id = i;
    }

    String getName( )
    {
        return name;
    }

    int getId( )
    {
        return id;
    }
    Eyes getEye( )
    {
        return e;
    }
}
```

```
class HumanBody
{
    HumanBody( )
    {
        System.out.println( "HumanBody Constructor Called" );
    }
}
class ENT extends HumanBody
{
    ENT( )
    {
        System.out.println( "ENT Constructor Called" );
    }
}
class Eyes extends ENT implements Serializable
{
    Eyes( )
    {
        System.out.println( "Eyes Constructor Called" );
    }
}

public class SerializationTest_5
{
    public static void main( String [] args )
    {
        Person subhash = new Person("Subhash", 18);
        Person sumesh  = new Person("Sumesh", 19);
        Person rajesh  = new Person("Rajesh", 20);

        try
        {
            FileOutputStream foutput = new FileOutputStream("person.txt");
            ObjectOutputStream ooutput = new ObjectOutputStream(foutput);

            ooutput.writeObject(subhash);
            ooutput.writeObject(sumesh);
            ooutput.writeObject(rajesh);

            ooutput.close();
        }
        catch( IOException ex )
        {
            ex.printStackTrace();
        }
        subhash = null;
        sumesh  = null;
        rajesh  = null;

        try
        {
```

```
            FileInputStream finput = new FileInputStream("person.txt");
            ObjectInputStream oinput = new ObjectInputStream(finput);
            Object one   = oinput.readObject();
            Object two   = oinput.readObject();
            Object three = oinput.readObject();

            oinput.close();
            subhash = (Person)one;
            sumesh  = (Person)two;
            rajesh  = (Person)three;
        }
        catch( Exception ex )
        {
            ex.printStackTrace();
        }


        System.out.println( subhash.getName());
        System.out.println( sumesh.getName());
        System.out.println( rajesh.getName());

        System.out.println( subhash.getId());
        System.out.println( sumesh.getId());
        System.out.println( rajesh.getId());

        System.out.println( subhash.getEye());
        System.out.println( sumesh.getEye());
        System.out.println( rajesh.getEye());
        System.out.println( rajesh.getEye());
    }
}
```



Output:

```
$ java SerializationTest_5
HumanBody Constructor Called
ENT Constructor Called
Eyes Constructor Called
Person Constructor Called
HumanBody Constructor Called
ENT Constructor Called
Eyes Constructor Called
Person Constructor Called
HumanBody Constructor Called
ENT Constructor Called
Eyes Constructor Called
Person Constructor Called
HumanBody Constructor Called
ENT Constructor Called
HumanBody Constructor Called
ENT Constructor Called
HumanBody Constructor Called
ENT Constructor Called
```

```
Subhash
Sumesh
Rajesh
18
19
20
Eyes@330bedb4
Eyes@2503dbd3
Eyes@4b67cf4d
$
```

How to write a string to a file ?

```
import java.io.*;
public class StringWritingTest
{
    public static void main( String [] args )
    {
        try
        {
            FileWriter fw = new FileWriter( "sample.txt" );
            fw.write("Subhash Programming Classes");
            fw.close( );
        }
        catch( IOException e )
        {
            e.printStackTrace();
        }
    }
}
```

Output:

(This program would create a file **sample.txt** and write the string **"Subhash Programming Classes"** in it)

How to list the contents of a directory ?

```
import java.io.*;

public class ListingDirectoryTest
{
    public static void main( String [] args )
    {
        int i;
        File f = new File("sampleDirectory");
        if( f.isDirectory( ) )
        {
            String [] list = f.list( );
            for( i = 0; i < list.length; i++ )
            {
                System.out.println( list[i] );
```

```
            {
                System.out.println( list[i] );
            }
        }
    }
}
```

Output:

(This program would read the contents of the directory and list all the files in it)

How to create a directory ?

```
import java.io.*;

public class CreatingDirectoryTest
{
    public static void main( String [] args )
    {
        File f = new File("chapter1");
        f.mkdir( );
    }
}
```

Output:

(This program creates a directory in your present working directory)

How to find my current path name of my file ?

```
import java.io.*;

public class AbsolutePathTest
{
    public static void main( String [] args )
    {
        File f = new File("chapter1");
        System.out.println( f.getAbsolutePath());
    }
}
```

Output:

```
$
/Users/subhash/Subhash/CJ25/SerializationAndInputOutput/chapter1
$
```

How to delete a file ?

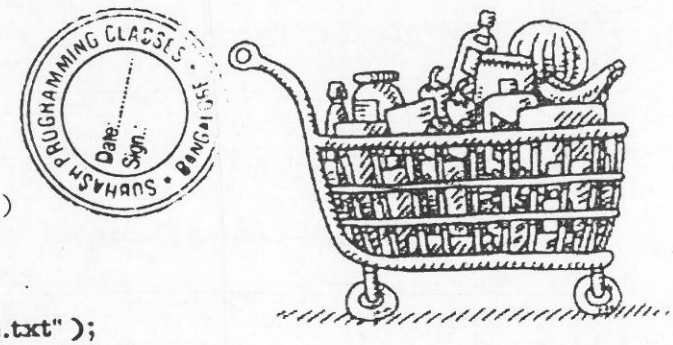
```
import java.io.*;
```

```
public class AbsolutePathTest
{
    public static void main( String [] args )
    {
        File f = new File("chapter1");
        System.out.println( f.delete( ));
    }
}
```

```
$
true
```

(This program deletes a file and returns **true / false** value)

Buffered Reader :




```
import java.io.*;

public class BufferedReaderTest
{
    public static void main( String [] args )
    {
        try
        {
            File file = new File( "sample.txt" );
            FileReader fr = new FileReader( file );
          BufferedReader br = new BufferedReader(fr);
            String line;
            while( (line = br.readLine( )) != null )
            {
                System.out.println( line );
            }
            br.close();
        }
        catch( Exception ex )
        {
            ex.printStackTrace( );
        }
    }
}
```

Output:

The above program reads the contents from the file "sample.txt" and prints it on the screen.

State True or False :

1. '**Serializable**' is a marker interface :
2. Instance variable that are declared 'transient' are not serialized.
3. If an instance variable is a reference to another object, then, even the reference variable class should implement '**Serializable**' :
4. **ObjectOutputStream** is a connection oriented stream :
5. **ObjectInputStream** is a chain oriented stream :
6. During de-serialization, the constructors for the deserialised object's constructor starts running :

Fill in the blanks :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

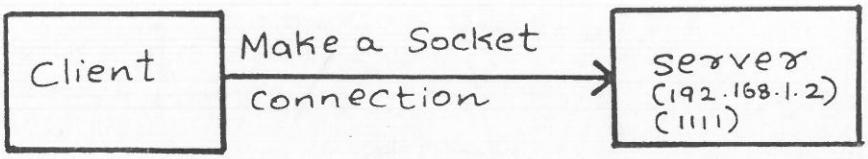
A ___ C	D E ___	G ___ I
J K ___	M ___ O	__Q__
S ___ U	VW ___	Y ___

1. '**ObjectInputStream**' is an example of _____ stream.
2. To avoid serializing an instance variable make it _____ .
3. '**FileInputStream**' is an example of _____ stream.
4. The **transient** variables are given a value of _____ for object references and respective _____ for primitives.

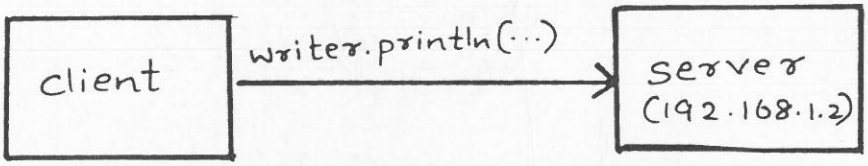
Networking, Threads & Synchronization

- A Socket is an object that represents the connection (relationship) between two machines.
- Three important steps to get the client working :
  - Establish initial connection between client and server.
  - Send messages to the server.
  - Receive messages from the server.
- A **port number** is a unique number to identify the server among multiple server on a single machine.
- An **IP address** identifies the machine on which the server runs.
- "127.0.0.1" is known as loop back address.
- Three important steps to get the server working :
  - Create a server site socket to bind to the **port**.
  - Accept the client request and create a new socket specifically to talk to the client via the new socket.
  - Receive messages from the client.
  - Send messages to the client.

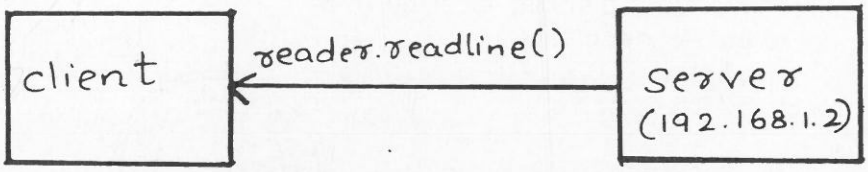
Connect :



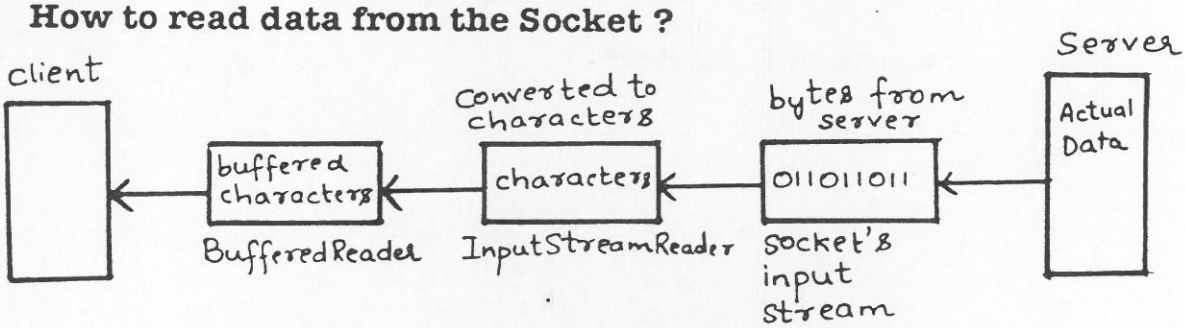
Send :



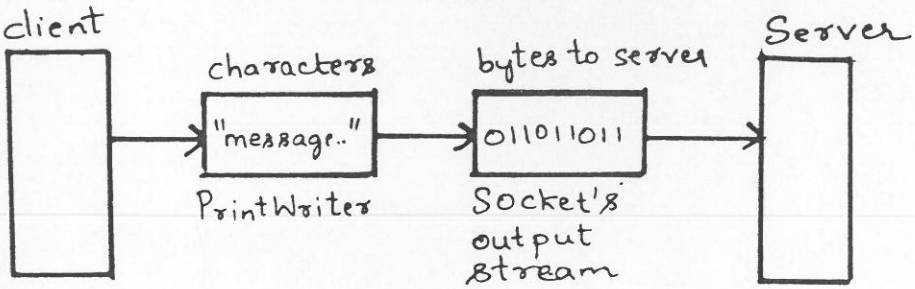
Receive :



How to read data from the Socket ?



How to write data to the Socket ?



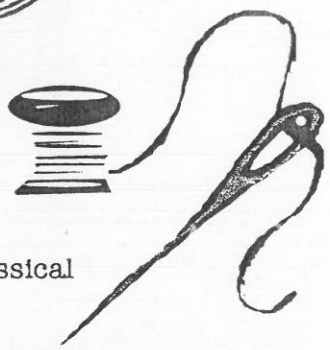
Client Code :

```
import java.io.*;
import java.net.*;

public class Client
{
    public static void go ( )
    {
        try
        {
            Socket clientSock = new Socket("127.0.0.1", 1111 );
            BufferedReader clientReader = new BufferedReader(new Input-
StreamReader(clientSock.getInputStream( )));
            String message = clientReader.readLine( );
            System.out.println( message );
            clientReader.close( );
        }
        catch( IOException e )
        {
            e.printStackTrace( );
        }
    }

    public static void main( String [] args )
    {
        Client c = new Client( );
        c.go( );
    }
}
```

Server Code :


```
import java.io.*;
import java.net.*;


public class Server
{
    public static void Start( )
    {
        try
        {
            ServerSocket serverSock = new ServerSocket( 1111 );
            while(true)
            {
                Socket connectionSock = serverSock.accept( );
                PrintWriter writer = new PrintWriter(connectionSock.getOutput-
Stream( ));
                writer.println( "Hi Baby" );
                writer.close( );
            }
        }
        catch( IOException e )
        {
            e.printStackTrace( );
        }
    }

    public static void main( String [] args )
    {
        Server s = new Server( );
        s.Start( );
    }
}
```

Threads :



- **Threads** are **light-weight process** that run within the address space of the main thread.
- One of the way **Multithreading** enables you to write efficient programs that make maximum use of the processing power available in the system by keeping the **idle time** to a minimum.
- Word Processors (Word, Pages) etc. are great classical examples that utilizes multithreading facility.
- Different states of a thread are as follows:
  - **Running** State
  - **Ready to Run** State
  - **Suspended** State
  - **Resumed** State
  - **Blocked** State
- Allocating CPU to a particular thread snatching it from other thread is known as **context switch**. This is done by pre-empting the running thread by an high priority thread.
- Java's multithreading system is built upon the **Thread** class and its companion interface - **Runnable**.
- **Thread** class encapsulates a thread of execution.
- When a Java Program starts up, one thread begins running immediately. This is usually called the '**main thread**' of your program, because it is the thread that is executed when your program begins.

Working with the main Thread:

```
public class CurrentThread
{
    public static void main( String[] args )
    {
        Thread t = Thread.currentThread( );
        System.out.println("Name of the current Thread is : " + t);

        t.setName("Subhash");
        System.out.println("Name of the current Thread after Name Change is : " + t);

        try
        {
            for( int i = 0; i < 5; i++ )
            {
                System.out.println(i);
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
```

```
        {
            System.out.println( "Main thread interrupted" );
        }


    }
}
```

```
$ java CurrentThread
Name of the current Thread is : Thread[main,5,main]
Name of the current Thread after Name Change is : Thread[Subhash,5,main]
0
1
2
3
4
$
```

How to create a new thread ?

1. Create a new class that implements a **Runnable** interface. You can construct a thread on any object that implements Runnable.
2. Implement the '**run( )**' method of the **Runnable** interface inside the new class just now created.
3. Create an **Thread** instance - say - 't' and call its '**start( )**' method which executes a call to **run( )**.

```
class NewThread implements Runnable
{
    Thread t;
    NewThread( )
    {
        t = new Thread( this, "I am a new Thread" );
        System.out.println("NewThread : " + t );
        t.start( );
    }
    public void run( )
    {
        try
        {
            for( int i = 0; i < 5; i++ )
            {
                System.out.println("New Thread : " + i);
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
        {
```

```
            e.printStackTrace();
        }
        System.out.println( "Exiting New Thread\n" );
    }
}

public class MainThread
{
    public static void main( String [] args )
    {
        new NewThread( );

        Thread t = Thread.currentThread( );
        t.setName ("Launching Thread" );
        System.out.println("Launching thread : " + t );
        try
        {
            for( int i = 0; i < 5; i++ )
            {
                System.out.println("Launching Thread : " + i );
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
        {
            e.printStackTrace( );
        }
        System.out.println( "Exiting Main Thread" );
    }
}

$
NewThread : Thread[I am a new Thread,5,main]
Launching thread : Thread[Launching Thread,5,main]
New Thread : 0
Launching Thread : 0
New Thread : 1
Launching Thread : 1
New Thread : 2
Launching Thread : 2
New Thread : 3
Launching Thread : 3
New Thread : 4
Launching Thread : 4
Exiting New Thread

Exiting Main Thread
$
```

How to create multiple threads ?

```
class NewThread implements Runnable
{
    Thread t;
    NewThread( String name )
    {
        t = new Thread( this, name );
        System.out.println("NewThread : " + t );
        t.start( );
    }
    public void run( )
    {
        try
        {
            for( int i = 0; i < 5; i++ )
            {
                System.out.println(t.getName() + " : " + i);
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }
        System.out.println( "Exiting " + t.getName() + " thread " );
    }
}

public class MainThread
{
    public static void main( String [] args )
    {
        new NewThread( "Subhash" );
        new NewThread( "Amitabh" );
        new NewThread( "Sachin" );

        Thread t = Thread.currentThread();
        t.setName("Launching Thread" );
        System.out.println("Launching thread : " + t );
        try
        {
            for( int i = 0; i < 5; i++ )
            {
                System.out.println("Launching Thread : " + i );
                Thread.sleep(1000);
            }
        }
        catch( InterruptedException e )
```

```
        {
            e.printStackTrace( );
        }
        System.out.println( "Exiting Main Thread" );
    }
}
```

How to synchronise multiple threads ?

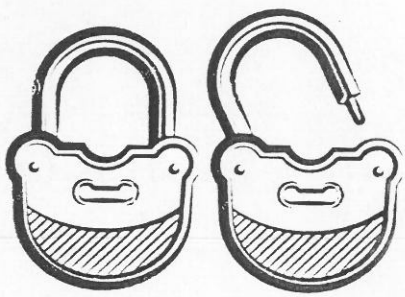
Without Sychronization:

```
class PrintMessage
{
    public void doPrint( String msg )
    {
        System.out.print( "[ " + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }
        System.out.println( "]" );
    }
}

class NewThread implements Runnable
{
    PrintMessage pm;
    String greetings;
    Thread t;

    NewThread( PrintMessage obj, String msg )
    {
        pm = obj;
        greetings = msg;

        t = new Thread( this );
        t.start( );
    }
    public void run( )
    {
        pm.doPrint( greetings );
    }
```

```
}

public class SynchronizationTest
{
    public static void main( String [] args )
    {
        PrintMessage pobj = new PrintMessage();

        NewThread subhash = new NewThread(pobj, "Hello" );
        NewThread rajkumar = new NewThread(pobj, "Namaskara" );
        NewThread rajinikanth = new NewThread( pobj, "Onakkam" );

        try
        {
            subhash.t.join( );
            rajkumar.t.join( );
            rajinikanth.t.join( );
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }

    }
}
```

```
$
[ Namaskara
[ Hello
[ Onakkam
]
]
]
$
```

With Sychronization:



```
class PrintMessage
{

    public synchronized void doPrint( String
msg )
    {
        System.out.print( "[ " + msg );
        try
        {
            Thread.sleep(1000);
        }
```

```
            catch( InterruptedException e )
            {
                e.printStackTrace();
            }
            System.out.println( "]" );
        }
}

class NewThread implements Runnable
{
    PrintMessage pm;
    String greetings;
    Thread t;

    NewThread( PrintMessage obj, String msg )
    {
        pm = obj;
        greetings = msg;

        t = new Thread( this );
        t.start( );
    }
    public void run( )
    {
        pm.doPrint( greetings );
    }
}

public class SynchronizationTest
{
    public static void main( String [] args )
    {
        PrintMessage pobj = new PrintMessage();

        NewThread subhash = new NewThread(pobj, "Hello" );
        NewThread rajkumar = new NewThread(pobj, "Namaskara" );
        NewThread rajinikanth = new NewThread( pobj, "Onakkam" );

        try
        {
            subhash.t.join( );
            rajkumar.t.join( );
            rajinikanth.t.join( );
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }
```

```
    }
}

$
[ Hello ]
[ Onakkam ]
[ Namaskara ]
$
```

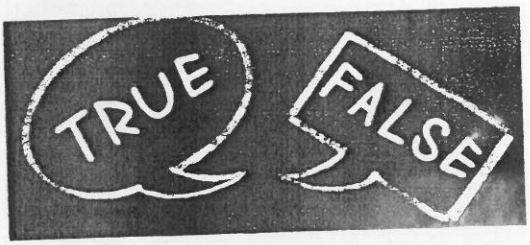
Another way to do Sychronization:

```
class PrintMessage
{

    public void doPrint( String msg )
    {
        System.out.print( "[ " + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }
        System.out.println( "]" );
    }
}

class NewThread implements Runnable
{
    PrintMessage pm;
    String greetings;
    Thread t;

    NewThread( PrintMessage obj, String msg )
    {
        pm = obj;
        greetings = msg;

        t = new Thread( this );
        t.start( );
    }
    public void run( )
    {
        synchronized(pm)
        {
            pm.doPrint( greetings );
        }
```

```
    }
}

public class SynchronizationTest
{
    public static void main( String [] args )
    {
        PrintMessage pobj = new PrintMessage();

        NewThread subhash = new NewThread(pobj, "Hello" );
        NewThread rajkumar = new NewThread(pobj, "Namaskara" );
        NewThread rajinikanth = new NewThread( pobj, "Onakkam" );

        try
        {
            subhash.t.join( );
            rajkumar.t.join( );
            rajinikanth.t.join( );
        }
        catch( InterruptedException e )
        {
            e.printStackTrace();
        }

    }
}
```


```
$
[ Hello]
[ Onakkam]
[ Namaskara]
$
```

State True or False :



1. Multiprocessing is different from Multi-threading :
2. A single server can be bound on multiple ports :
3. The server creates another socket to respond to the client along with the server socket :
4. A thread is known as a light-weight process :