Classes

We have seen that Java is an Object Oriented Language. It means that, in Java, everything is defined in terms of classes and objects. Classes are the core of Java. Classes are essentially the blueprints of objects.

What is a class made up of? In general, classes have variables (data) and set of methods (functions) that it performs.

The syntax of a class is shown in the following code.

```
class classname
{
  data type instance_variable1;
  data type instance_variable2;
  ... data type instance_variableN;

  data type method1(parameters)
  {
    Body of the method
  }
  data type method2(parameters)
  {
    Body of the method
  }
  ... data type methodN(parameters)
  {
    Body of the method
  }
}
```

In the above snippet, data type defines the variable type in case of variables. In case of methods the data type defines the type of data which the method (function) returns.

The following code demonstrates how a class is declared.

```
class Measurements
{
  int length, breadth, height;
  void computeArea(int length, int breadth)
  {
    int area = length * breadth;
    System.out.println(area);
  }
  void computeVolume(int length, int breadth, int height)
  {
    int volume = length * breadth * height;
    System.out.println(volume);
  }
}
```

In the above code, we have defined a class called Measurements. Within the class, we have defined three variables which are called class or instance variables. Also, we have defined two methods, computeArea(), computeVolume(). They are defined so that they compute and print the area and volume for the dimensions provided.

Objects

Objects can simply be defined as snapshots of classes. Classes provide the most structure ,or the blueprints for objects and objects are those that have particular properties(variables) and methods(functions) that are defined by the class at any instance.

For example, if we consider a Car to be a class, BMW can be one object and Mercedes can be another. However, both of them have the same set of properties (such as color, length, height etc.) and functionalities (such as steering, braking etc.) that are defined by the class Car.

Let us now see how to define objects for any class.

Object Creation

Objects are created for a class using the new keyword. Once we create an object for any particular class, we can access the members of the class using the dot operator.

The following code demonstrates how objects are created for a class.

```
class Measurements
{
  int length, breadth, height;
  void computeArea(int length, int breadth)
  {
    int area = length * breadth;
    System.out.println(area);
  }
  void computeVolume(int length, int breadth, int height)
  {
    int volume = length * breadth * height;
    System.out.println(volume);
  }
}
class Demo
{
  public static void main(String args[])
  {
    Measurements m = new Measurements();
    m.computeArea(5,5);
  }
}
```

In the above code, inside the 'Demo' class, we have created an object for the "Measurements" class using the new keyword (m is the object). Further, the methods (computeArea) in "Measurements" class are accessed using the dot operator (m.computeArea).

IMPORTANT: On compiling the above code, we get two class files (Measurements.class and Demo.class). We need to run the class file 'Demo' to get the output when we run this code on our systems (java Demo). This is because the class Demo contains the main() function.

Try this yourself by creating another object and computing the volume with that object, by giving an additional argument "height" to the computeVolume() method. Ensure that you are comfortable with the working of classes and objects before you move on to the next set of topics.

Test Yourself

Q1. How to access any member of class from object?

Show/Hide Answer

Ans. By using dot operator.

More Exercises

1. Create a class "Student" which has the following properties and functionalities:
    - Name
    - Roll number
    - Math score
    - Physics score
    - Chemistry score
    - Total score
    - Create a method 'show_Total' which accept 5 parameters (name, roll and 3 subjects scores) and display the total of three scores. The output should look like as below:

```
Name : Bill Gates
Roll Number : 09331A0786
Math : 40
Physics : 30
Chemistry : 20
Total : 90
```

    - Create a student object in main method. Using this object in main method, call 'show_Total' method to display total of scores.

Class Members

We have studied the structure of a class and its composition. The class variables and the methods together form the Class Members. Let us look at these in more detail.

Class Variables

Class variables are the variables which are visible over the entire class and accessible to all the methods in the class. When we create an object for a class, a set of instance variables are created for that object. They are accessed by the object using the dot operator, as we have seen in the previous tutorial.

Methods

Methods are generally used to achieve a particular task. We have seen the use of methods like computeArea() and computeVolume() in the previous topic. Both of them are used for a specific purpose. Further, we have also seen how arguments are passed to these methods.

There is an important type of method called the Constructor. It is method-like because, it has the same structure as a method. The constructor is called automatically when an object is created. Therefore, the constructor initializes an object automatically and immediately after its creation.

Remember: 1) The constructor does not have a return type like that of methods. It does not even return a void type. It returns implicitly the same class type.
2) Constructors have the same name as that of the class. This is how the code differentiates between the constructor and any other method.

The following code demonstrates how constructors are used using the same example which we have seen previously, but with some modification.

```
class Measurements
{
  int length, breadth, height;
  Measurements()
  {
    System.out.println("Constructing the class");
    length = 5;
    breadth = 2;
    height = 3;
  }
  void computeArea()
  {
    int area = length * breadth;
    System.out.println(area);
  }
  void computeVolume()
  {
    int volume = length * breadth * height;
    System.out.println(volume);
  }
}
class Demo
{
  public static void main(String args[])
  {
    Measurements m = new Measurements();
    m.computeArea();
    m.computeVolume();
  }
}
```

In the above code, we have created a class 'Measurements' which has three class variables; length, breadth and height. Also, we have defined two methods; computeArea() and computeVolume(). We have also defined a constructor which has the same name as that of the class. We have initialized the class variables inside the constructor and also printed a statement in order to show that the constructor is automatically called upon creation of the object.

Parameterizing the Constructor

Just like we have parameters for methods, we have used parameters for this constructor. The following code demonstrates how a parameterized constructor looks like and how it is used.

```
class Measurements
{
  Measurements(int l, int b, int h)
  {
    System.out.println("Constructing the class");
    int length = l;
    int breadth = b;
    int height = h;
    int volume = length * breadth * height;
    System.out.println(volume);
  }
}


class Demo
{
  public static void main(String args[])
  {
    Measurements m = new Measurements(5,3,2);
  }
}
```

Try to understand how the above code works. Try this yourself. Also, try to change as many things as possible and play around with the code to get a better understanding of how the constructor works.

More Exercises

1. Create a class "Student" which has the following properties and functionalities:
    - Name
    - Roll number
    - Math score
    - Physics score
    - Chemistry score
    - Total score
    - Constructor which accepts 5 parameters: Name, Roll number, and 3 subjects' scores. Assign these values to the above created class variables.
    - Create a method 'show_Total' which accept 3 parameters (subjects scores) and display the total of three scores.
    - Create a student object in main method passing the corresponding parameters of constructor.
    - Using the object in main method, call 'show_Total' method to display total of scores.
2. Create a class as Puppy.Overload it to pass the name of the puppy.Use the set and get methods to find the puppy's age .
3. Create a class as Employee with name,age,designation and salary of the employee.Use methods to initialize and declare the age,designation and salary of the employee.Use another class as EmployeeTest to display the details of an employee.
4. Write a program to show Class and Object initialization showing the Object Oriented concepts in Java.

Method Overloading

If a class has multiple methods by same name but different parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

1)Example of Method Overloading by changing the no. of arguments:

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
class Calculation{
 void sum(int a,int b){System.out.println(a+b);}
 void sum(int a,int b,int c){System.out.println(a+b+c);}
  public static void main(String args[]){
  Calculation obj=new Calculation();
  obj.sum(10,10,10);
  obj.sum(20,20);
 }
}
Output:30
       40
```

2)Example of Method Overloading by changing data type of argument:

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
class Calculation{
  void sum(int a,int b){System.out.println(a+b);}

 void sum(double a,double b){System.out.println(a+b);}

  public static void main(String args[]){
  Calculation obj=new Calculation();
  obj.sum(10.5,10.5);
```

```
  obj.sum(20,20);


  }
}
Output:21.0
```

Test Yourself

Q1. Why Method Overloaing is not possible by changing the return type of method?

Show/Hide Answer

Ans.In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur: because there was problem:

```
  class Calculation{
  int sum(int a,int b)
  {System.out.println(a+b);}
  double sum(int a,int b)
  {System.out.println(a+b);}


  public static void main(String args[]){
  Calculation obj=new Calculation();
  int result=obj.sum(20,20); //Compile Time Error


  }
}
int result=obj.sum(20,20); //Here how can java determine which sum() method should be called.
```

Q2.Can we overload main() method?

Show/Hide Answer

Ans.Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

```
class Simple{
  public static void main(int a){
  System.out.println(a);
  }


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

```
  System.out.println("main() method invoked");
  main(10);
  }
}
Output:main() method invoked
       10
```

More EXERCISES

1. Write a program using method overloading,find the volume of three rooms-a,b,c.
2. Write a program to find the area of a rectangle whose length and breadth are:10,20 and 10.5 ,20.5.
3. Construct a code that gives the height of the current house and height of a building to be constructed using method and constructor overloading.

---

static keyword

The static keyword is used in java for memory management. The static keyword is applied with variables, methods, blocks and nested class. It belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block

1) static variable

If you declare any variable as static, it is known static variable.

1. The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
2. The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable
1)It makes your program memory efficient .

Understanding problem without static variable

```
  class Student{
```

```
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 studentsin my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

2)static property is shared to all objects.

Example of static variable

```
//Program of static variable
class Student{
   int rollno;
   String name;
   static String college ="ITS";

   Student(int r,String n){
   rollno = r;
   name = n;
   }
 void display (){System.out.println(rollno+" "+name+" "+college);}

 public static void main(String args[]){
 Student s1 = new Student (111,"Karan");
 Student s2 = new Student (222,"Aryan");

 s1.display();
 s2.display();
 }
}
Output:111 Karan ITS
       222 Aryan ITS
```

Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in theconstructor. Since instance variable gets the memory at the time of object creation, each

object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

```
class Counter{
int count=0;//will get memory when instance is created

Counter(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();

}}
Output:1
       1
       1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
  class Counter{
static int count=0;//will get memory only once and retain its value

Counter(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter c1=new Counter();
Counter c2=new Counter();
```

```
Counter c3=new Counter();


}}
Output:1
       2
       3
```

2) static method

If you apply static keyword with any method, it is known as static method

1. A static method belongs to the class rather than object of a class.
2. A static method can be invoked without the need for creating an instance of a class.
3. static method can access static data member and can change the value of it. Example of static method

```
//Program of changing the common property of all objects(static field).

class Student{
     int rollno;
     String name;
     static String college = "ITS";

     static void change(){
     college = "BBDIT";
     }

     Student(int r, String n){
     rollno = r;
     name = n;
     }

     void display (){System.out.println(rollno+" "+name+" "+college);}

    public static void main(String args[]){
    Student.change();

    Student s1 = new Student (111,"Karan");
```

```
    Student s2 = new Student (222,"Aryan");
    Student s3 = new Student (333,"Sonoo");

    s1.display();
    s2.display();
    s3.display();
    }
}
Output:111 Karan BBDIT
       222 Aryan BBDIT
       333 Sonoo BBDIT
```

Another example of static method that performs normal calculation

```
//Program to get cube of a given number by static method

class Calculate{
  static int cube(int x){
  return x*x*x;
  }

  public static void main(String args[]){
  int result=Calculate.cube(5);
  System.out.println(result);
  }
}
Output:125
```

Restrictions for static method:

There are two main restrictions for the static method. They are: 1)The static method can not use non static data member or call non-static method directly. 2)this and super cannot be used in static context.

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
```

```
 }
}
Output:Compile Time Error
```

3)static block

Is used to initialize the static data member. It is executed before main method at the time of classloading.

Example of static block

```
 class A{
  static{System.out.println("static block is invoked");
  }
   public static void main(String args[]){
   System.out.println("Hello main");
  }
}
Output:static block is invoked
       Hello main
```

Test Yourself

Q1. why main method is static?

Show/Hide Answer

Ans.because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

Q2.)Can we execute a program without main() method?

Show/Hide Answer

Ans.Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
class A{
  static{
  System.out.println("static block is invoked");
  System.exit(0);
  }
}
```

```
Output:static block is invoked (if not JDK7)
```

Inheritance

Inheritance is one of the basic principles of OOP. Using Inheritance, we can create hierarchical classifications. If defined so, one class may inherit the variables and methods of another class. The class that is inherited is called the super class, and the class that does inheriting is called sub-class.

Basics of Inheritance

A class can inherit another class using the extends keyword. By inheritance, a class can inherit all the properties (variables & methods) of its parent. But, there could be some possible cases, where a part of properties can be restricted to the base class alone. This can be done usingaccess specifiers.

For the scope of this course, we shall discuss the following access specifiers.

- Private Access Specifier: If we declare any variable or method to be private, it is accessible only by the members of its own class.
- Public Access Specifier: If we declare a class member to be public, it is accessible by all the classes that inherit the base class either directly or indirectly.
- Default Access Specifier: There is also a third type of access specifier called default. This is the access specifier for any property in the class which does not have any access specifier defined (hence the name default). It has package visibility. That means that the class members declared as default, will be accessible only in those classes that inherit the base class directly or indirectly and also in the same package as that of the base class.

The following code demonstrates how inheritance is achieved using the extends keyword

```
class A
{
  public int a = 5; // public variable. Accessible in child class which
extends class A.
  private int c = 6; // private variable. Cannot be accessed outside
class A.
  void superclass()
  {
    System.out.println("Super class" + " " + "value of a is " + a);
    System.out.println("Super class" + " " + "value of c is " + c);
  }
}
class B extends A
{
```

```
  int b = 7;
  int a = 8; //obtained from the parent class. Note that, c is private
to class A and is not available in class B
  void subclass()
  {
    System.out.println("Sub class" + " " + "value of b is " + b);
    System.out.println("Sub class" + " " + "value of a is " + a);
  }
}
class Demo
{
  public static void main(String args[])
  {
    A a1 = new A();
    B b1 = new B();
    a1.superclass();
    b1.subclass();
    b1.superclass();
  }
}
```

In the code above, class A is the parent of class B and is inherited using the 'extends' keyword. In class Demo, we have created objects each for classes A and B. Note here that, we have called a method of class A using an object of class B. This is possible because the public methods and public variables of class A are inherited into child class B.

Remember that, even methods can have access specifiers like variables. For example, the code above would not work if we defined the method in class A as private void superclass(). Try this out yourself. Also, test concepts of inheritance by using different scenarios and observe the outputs.

Test Yourself

Q1. Which of the keyword must be used to inherit a class?

Show/Hide Answer

Ans. extends.

More EXERCISES

1. Create a class 'Father' with following properties:
    - Name
    - Surname
    - A method 'show_Hair' which displays the string "has bald head"
        - Create another class 'Son' which extends 'Father' class with following properties:
            - Name
            - A method 'show_Eye_Sight' which displays the string "is blind"
            - In main method create 'Son' object and call 'show_Hair' and 'show_Eye_Sight' methods.

2. Create an Interface float and implement its properties and methods in boat, ship classes
3. Write a program using inheritance to find the sum and difference of two integers given as input by the user.
4. Write a program using inheritance display the coordinates of a point in 3D

Super keyword

The super is a reference variable that is used to refer immediate parent class object.Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of super Keyword:

1.super is used to refer to the immediate parent class instance variable
2.super() is used to invoke immediate parent class constructor.
3.super is used to invoke immediate parent class method.
a) super is used to refer to the immediate parent class instance variable

```
class Car{
 int speed=50;
              }
     class Etios extends Car{
     int speed=100;
     void display()     {
     System.out.println(super.speed);//will print speed of Car now
                        }
     public static void main(String args[]){
     Etios b=new Etios();
     b.display();       }
                        }
         Output:50
```

b)The super keyword can also be used to invoke the parent class constructor as given below:

```
class Car{


Car()
         {
     System.out.println("Car is created");}
```

```
    }
class Etios extends Car{
Etios(){
super();//will invoke parent class constructor
System.out.println("etios is created");
      }
public static void main(String args[]){
Etios b=new Etios(); }
      }
Output:Car is created
Etios is created
```

Note: super() is added in each class constructor automatically by compiler. If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

Another example of super keyword where super() is provided by the compiler implicitly:

```
class Vehicle{
Vehicle(){
     System.out.println("Vehicle is created");}
          }
     class Bike extends Vehicle{
     int speed;
     Bike(int speed){
     this.speed=speed;
     System.out.println(speed);
          }
     public static void main(String args[]){
     Bike b=new Bike(10);
          }
          }
Output:Vehicle is created
       10
```

c) super is used to invoke immediate parent class method

```
class Person{
void message(){
     System.out.println("welcome");}
```

```
                }
    class Student extends Person{
    void message(){
    System.out.println("welcome to java");}
    void display(){
    message();//will invoke current class message() method
    super.message();//will invoke parent class message() method
                }
    public static void main(String args[]){
    Student s=new Student();
    s.display();  }
                }
    Output:welcome to java
     welcome
```

Test Yourself

Q1. What is the difference between super and this keyword?

Show/Hide Answer

Ans. super() is used to call super class constructor, whereas this() used to call constructors in the same class.

Abstract Classes

Abstract classes are classes that have at least one abstract method. An abstract method is the one that is declared, but does not have any definition (or is unimplemented).

Abstract classes cannot be instantiated. That means, objects cannot be created for abstract classes. However, a concrete class can extend the abstract class and provide the implementation (or definition) for the abstract method. This concrete class can then be instantiated (an object can be created for this concrete class).

Abstract methods are used when two or more subclasses are expected to fulfil a similar role in different ways through different implementations. Usually, these subclasses extend the same 'abstract class' and provide different implementations for the abstract methods. This will ensure that the name of the method will be the same in the subclasses, but the function it performs will be different.

The following code demonstrates the concept of abstract classes.

```
abstract class Person
{
  public abstract void getOccupation();
  public void getCity()
  {
   System.out.println("Working in Gurgaon");
  }
}
class Employee1 extends Person
{
  public void getOccupation()
  {
   System.out.println("Working at Internshala");
  }
}
class Employee2 extends Person
{
  public void getOccupation()
  {
   System.out.println("Working in the Education Domain");
  }
}
class Demo
{
  public static void main(String args[])
  {
    Employee1 e1 = new Employee1();
    e1.getOccupation();
    e1.getCity();
    Employee2 e2 = new Employee2();
    e2.getOccupation();
    e2.getCity();
  }
}
```

In the code above, the class 'Person' has an unimplemented method getOccupation. So, the class 'Person' is declared as an abstract class. Now, note that we cannot create an object for the class 'Person'. Doing so would result in an error. Now, the class 'Employee' is inherited from the class 'Person'. The unimplemented method getOccupation has been implemented in the child class 'Employer'. This is how abstract classes are used. Try it out yourself

Why do we use Abstract Class?

The advantage of an abstract class is that it enforces certain hierarchies or standards for all the subclasses. Remember that, if any class extends an abstract class, then the inherited class MUST provide implementation for all the unimplemented methods of the abstract class. Not doing so would throw an error. For example, in the example provided, the class 'Person' is abstract and has an unimplemented method 'getOccupation()'. This abstract class is extended by the classes 'Employee1' and 'Employee2'. Now, if we do not provide the implementation for the method 'getOccupation()' in either of these classes, then the code would throw an error.

Therefore, we are essentially forcing any class that extends the abstract class to use the method getOccupation() and provide an implementation for it. This ensures that the standard is maintained through out the code (every class provides implementation for method 'getOccupation()'). This is abstract classes are used.

Interfaces

Interfaces define a standard and public way of specifying the behaviour of classes. All the methods of an interface are abstract. It allows class inheritance, so as to implement common behaviours.

The main reasons for using interfaces is to bring out an object's functionality without revealing its implementation. This helps maintain uniformity and standard throughout the code.

The following code demonstrates the concept of interfaces.

```
interface  Person
{
  public abstract void getOccupation();
  public abstract void getCity();
}
class Employee1 implements Person
{
  public void getOccupation()
  {
   System.out.println("Working at Internshala");
  }
  public void getCity()
  {
   System.out.println("Working in Gurgaon");
  }
}
class Employee2 implements Person
{
  public void getOccupation()
  {
   System.out.println("Working in the Education Domain");
  }
  public void getCity()
  {
   System.out.println("Working in Delhi");
  }
}
class Demo
{
  public static void main(String args[])
  {
    Employee1 e1 = new Employee1();
    e1.getOccupation();
    e1.getCity();
    Employee2 e2 = new Employee2();
    e2.getOccupation();
    e2.getCity();
```

```
    }
}
```

Interfaces Vs Abstract Classes

Following are the main differences between abstract classes and interfaces.

- In an interface, all the methods are abstract. But in case of an abstract class, it is not necessary that all the methods are abstract. A class is called abstract if at least one method is abstract.
- Interfaces have no direct inherited relationship with any particular class, they are defined independently. This means an interface can never be inherited from another class, while an abstract class can.
- Interfaces are implemented using the implements keyword whereas Abstract classes are extended using extends keyword.
- Interface is absolutely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists. This we can define the main function in an abstract class and create an object for itself, but this is not possible in the case of an interface.
- In comparison with Java abstract classes, Java interfaces are slow.

We will get a more clear picture about abstract classes and interfaces when we use this in practice. Meanwhile, try to implement both abstract classes and interfaces in the box and observer the outputs.

Test Yourself

Q1. Can abstract class have Constructor?

Show/Hide Answer

Ans. Yes.

More EXERCISES

1. Create a abstract class 'Car' which has following properties:
    - An abstract method 'show_Color'.
    - A method 'show_wheels' which prints number of wheels.
    - An abstract method 'show_Speed'.
    - An abstract method 'show_Weight'.
        - Create a class 'Mercedes' which extends 'Car' class
        - A method which implements the abstract method 'show_Color'. Print color of car.
        - A method which implements the abstract method 'show_Speed'. Print speed of the car.
        - Create an object of 'Mercedes' and call all the methods.
        - Create a class 'Alto' which extends 'Car' class

- A method which implements the abstract method 'show_Color'. Print color of car.
- A method which implements the abstract method 'show_Weight'. Print car weight in KG.
- Create an object of 'Alto' and call all the methods.

This will give errors, think about why you are getting these errors and fix them.

2. Create a class 'calculate' and apply addition and division operations for two, three and four numbers by changing the method.

Exception Handling

Scenarios where exception may occur

Exception handling provides the mechanism to handle the runtime errors so that normal flow of the application can be maintained.

Exception

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime

Exception Handling

Exception Handling is a mechanism to handle runtime errors

Advantage of Exception Handling

The core advantage of exception handling is that normal flow of the application is maintained. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a case:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;
```

Suppose there are 5 statements in your program and there occurs an exception at statement 3, rest of the code will not be executed i.e. statement 4 and 5 will not run. If we perform exception handling, rest of the exception will be executed. That is why we use exception handling.

Types of Exception:

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Common scenarios of Exception Handling where exceptions may occur:

There are given some scenarios where unchecked exceptions can occur. They are as follows: 1) Scenario where ArithmeticException occurs If we divide any number by zero, there occurs an ArithmeticException. int a=50/0;//ArithmeticException

2) Scenario where NullPointerException occurs If we have null value in any variable, performing any operation by the variable occurs an NullPointerException. String s=null; System.out.println(s.length());//NullPointerException

3) Scenario where NumberFormatException occurs The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException. String s="abc"; int i=Integer.parseInt(s);//NumberFormatException

4) Scenario where ArrayIndexOutOfBoundsException occurs If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below: int a[]=new int[5]; a[10]=50; //ArrayIndexOutOfBoundsException

Use of try-catch block in Exception handling: Five keywords used in Exception handling:

- try
- catch
- finally
- throw
- throws

try block:

Enclose the code that might throw an exception in try block. It must be used within the method and must be followed by either catch or finally block.

Syntax of try with catch block:

```
try{ ... }catch(Exception_class_Name reference){}
```

Syntax of try with finally block:

```
try{ ... }finally{}
```

catch block:

Catch block is used to handle the Exception. It must be used after the try block.

Problem without exception handling:

```
class Simple{
 public static void main(String args[]){
     int data=50/0;
     System.out.println("rest of the code...");
}
}
Output:Exception in thread main java.lang.ArithmeticException:/ by
  zero
```

As displayed in the above example, rest of the code is not executed i.e. rest of the code... statement is not printed.

Test Yourself

Q1.What happens behind the code int a=50/0;?
Show/Hide Answer

Ans.The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks: 1.Prints out exception description. 2.Prints the stack trace (Hierarchy of methods where the exception occurred). 3.Causes the program to terminate. But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

Solution by exception handling:

```
 class Simple{
  public static void main(String args[]){
   try{
     int data=50/0;
      }catch(ArithmeticException e){System.out.println(e);}
        System.out.println("rest of the code...");
}
}
Output:Exception in thread main java.lang.ArithmeticException:/ by
```

```
  zero

       rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

String Handling

String Handling provides a lot of concepts that can be performed on a string such as concatenating string, comparing string, substring etc.

String

Generally string is a sequence of characters. But in java, string is an object. String class is used to create string object. How to create String object? There are two ways to create String object:
a)By string literal
b)By new keyword

1) String literal

String literal is created by double quote.For Example: String s="Hello";

```
String s1="Welcome";
String s2="Welcome";//no new object will be created
```

In the above example only one object will be created.First time JVM will find no string object with the name "Welcome",so it will create a newobject.Second time it will find the string with the name "Welcome",so it will not create new object whether will return the reference to the same instance.

2) By new keyword

String s=new String("Welcome");//creates two objects and one reference variable

Concatenating Strings:

The String class includes a method for concatenating two strings: string1.concat(string2); This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals, as in: "My name is ".concat("Zara");

Strings are more commonly concatenated with the + operator, as in: "Hello," + " world" + "!" which results in: "Hello, world!"

Methods of String class in Java

java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting strings etc. Let's see the important methods of String class.

1)public boolean equals(Object anObject)	Compares this string to the specified object.
2)public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)public String concat(String str)	Concatenates the specified string to the end of this string.
4)public int compareTo(String str)	Compares two strings and returns int
5)public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)public String substring(int beginIndex)	Returns a new string that is a substring of this string.
7)public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)public String toUpperCase()	Converts all of the characters in this String to upper case
9)public String toLowerCase()	Converts all of the characters in this String to lower case.
10)public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
11)public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)public char charAt(int index)	Returns the char value at the specified index.
14)public int length()	Returns the length of this string.
15)public String intern()	Returns a canonical representation for the string object.
16)public byte[] getBytes()	Converts string into byte array.
17)public char[] toCharArray()	Converts string into char array.
18)public static String valueOf(int i)	converts the int into String.
19)public static String valueOf(long i)	converts the long into String.
20)public static String valueOf(float i)	converts the float into String.
21)public static String valueOf(double i)	converts the double into String.
22)public static String valueOf(boolean i)	converts the boolean into String.
23)public static String valueOf(char i)	converts the char into String.
24)public static String valueOf(char[] i)	converts the char array into String.
25)public static String valueOf(Object obj)	converts the Object into String.
26)public void replaceAll(String firstString,String secondString)	Changes the firstString with secondString.

Test Yourself

Q1. Which constructor is used to create an empty String object?

Show/Hide Answer

Ans. String().

More EXERCISES

1. Write a program using the System.currentTimeMillis() method generate the time taken to concatenate strings using + operator and time taken to concatenate strings using String Buffer(append()).
2. Write a program to Reverse the palindrome: Dot saw I was Tod
3. Write a program to convert a string to its lowercase.
4. Write a program to sheck whether two strings are equal or not.If not then compare the strings.

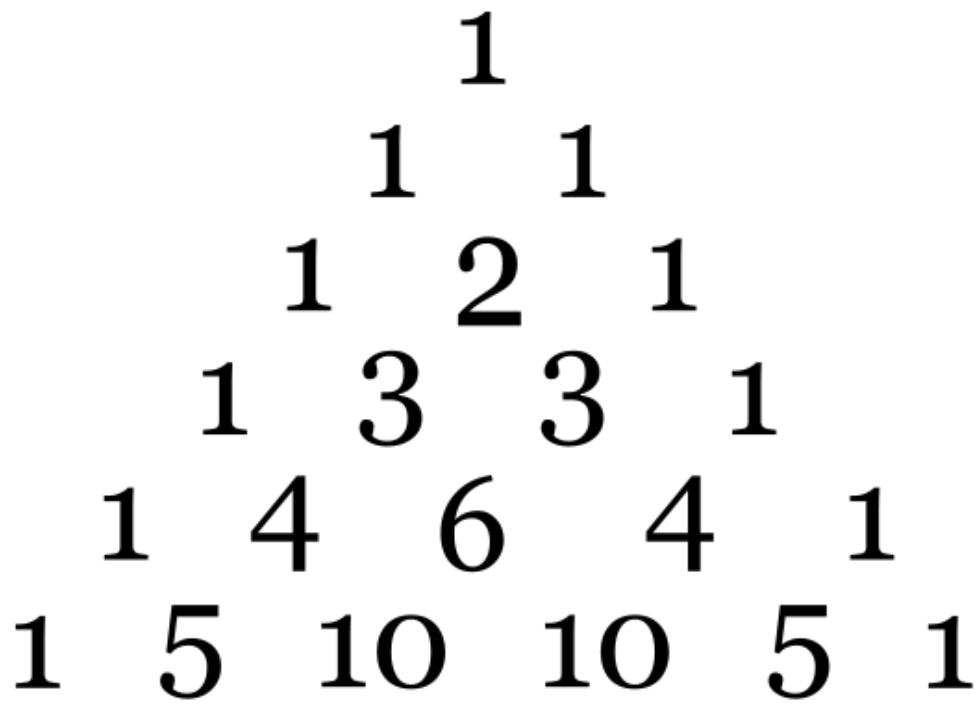
Algorithms

An Algorithm is an effective method that can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function.Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.

Some of the basic Algorithms are:-

Pascal Triangle

A system of numbers arranged in rows resembling a triangle with each number in the triangle is the sum of the two directly above it



Problem Statement:-

Create a pascal triangle with n number of rows where n is the input taken from the user

Solution:-

Click here to download solution.

Bubble Sort

Bubble sort is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. This algorithm is usually used when the list is short.

The algorithm for Bubble Sort is

```
procedure bubbleSort( A : list of sortable items )
   n = length(A)
   repeat
     swapped = false
     for i = 1 to n-1 inclusive do
       /* if this pair is out of order */
       if A[i-1] > A[i] then
         /* swap them and remember something changed */
         swap( A[i-1], A[i] )
         swapped = true
```

```
      end if
    end for
  until not swapped
end procedure
```

Problem Statement:-

Sort the array {56, 12, 91, 2, 0, 112, 4, 73, 81} using bubble sort

Solution:-

Click here to download solution.

Binary Search

In computer science, a binary search algorithm finds the position of a specified input value (the search "key") within an array sorted by key value.For binary search, the array should be arranged in ascending or descending order.

> In each step, the algorithm compares the search key value with the key value of the middle element of the array.

- If the keys match, then a matching element has been found and its index, or position, is returned.
- Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right.
- If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

The Recursive algorithm for Binary Search is

```
int binary_search(int A[], int key, int imin, int imax)
{
  // test if array is empty
  if (imax < imin)
    // set is empty, so return value showing not found
    return KEY_NOT_FOUND;
  else
    {
      // calculate midpoint to cut set in half
      int imid = midpoint(imin, imax);

      // three-way comparison
      if (A[imid] > key)
        // key is in lower subset
```

```
        return binary_search(A, key, imin, imid - 1);
      else if (A[imid] < key)
        // key is in upper subset
        return binary_search(A, key, imid + 1, imax);
      else
        // key has been found
        return imid;
    }
}
```

Problem Statement:-

Search a number in the list {10,145,24,50,176,900,999,111} using binary search

Solution:-

Click here to download solution.

Examples

Some of the basic examples used in Java are as follows:-

Armstrong Number

An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3**3 + 7**3 + 1**3 = 371. Click here to download.

Diamond Shape using Asterisks

With the help of nested for loops we can create a diamond shape of asterisks. Click here to download.

Factorial of a Number

The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example, 5!=5*4*3*2*1=120. Click here to download.

Fibonacci Series

Fibonacci Series is the series in which the first two numbers are either 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two. Click here to download.

Palindrome of a Number

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward. Click here to download.

Reverse of a Number

This program will print out the reverse of any number input by user. Click here to download.

Sum Of Digits

This program will give the sum of each individual digits of a number.For example, 345 will give output as 12. Click here to download.

Sum Of Prime Numbers

This program will print out the sum of n prime numbers where n is the input given by the user. Click here to download.

Multiplication Of a Number

This program will generate the multiplication table of any number input by the user. Click here to download.