Classes and objects

Class

- A class is an abstract concept. It is a collection of variables and methods.
- These variables and methods are called as “member variables” of that class.
- A class is generally a representation of any physical entity that exists in the real world eg: students, employees, Cars, Animals etc.
- When a class is created in java no memeory is assigned to it.
- The memory is assigned to an object that we create of that class.
- See the example below:

WAP to create a class called Circle, it should have three methods namely accept radius, calculate area

```
import java.util.*;
class Circle
{
private float r,area;
public void accept(float
x)
{
  r=x;
}
public void calculate()
{
  area=3.14f*r*r;
}
public void display()
{

System.out.println("Area=
"+area);
}
}
```

```
class Main
{
public static void main(String args[])
{
  float x;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter Radius:");
  x=sc.nextFloat();
  Circle c=new Circle();
  c.accept(x);
  c.calculate();
  c.display();
}
}
```

Access control

- Public: Those fields that are declared public are accessible by all classes in the same or different package.
- Private: They cannot be accessed by any methods except the ones in the same class.
- Protected: they can be accessed by every methods except for methods in non sub classes of different package
- Private protected: they can be accessed only by sub classes that can be of same package or other package.
- Default: they are accessible only by the methods of the same class, subclasses in same and different packages.

Access control

	Public	Protected	Default	Private Protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Sub class in the same package	Yes	Yes	Yes	Yes	No
Other classes in the same package	Yes	Yes	Yes	No	No
Subclasses in the other packages	Yes	Yes	No	Yes	No
other classes in the other packages	Yes	No	No	No	No

Tips to remember the previous table

- Protected :- In the same package, it can be accessed whether subclass or not, but in another package, it has to be a subclass.
- Default:- In the same package it is accessible whether subclass or not but in another package it is not possible even if it is a subclass.
- Private protected:- As long as it's a subclass, whether in the same or another package.

Constructors

- Constructors are used to create objects of a class. Basically a constructor initializes the values of the member variables in the class.
- Constructor is a special member used to initialize the members of the class.
- It is automatically invoked when the object is created.
- A constructor takes the responsibility of initializing the member variables of the class.
- The name of the constructor is same as that of the class.
- There are some more features of constructors below.

Features of constructor

- Name of the constructor has to be same as of the class.
- It cannot have return type not even void.
- They are automatically invoked.
- There can be more than one constructor, with different parameter list in a class.(called as constructor overloading)
- They should always be in the public/default/protected visibility of a class.

Types of constructors

- Parameterized constructors: These constructors accept argumetns at the time of object creation. The values that tare received as arguments are used by the constructor to initialize the member variables of the object of that class.
- Default constructors: These constructors do not accept any arguments at the time of object creation. So the initialization of member variables can take place in either of the two ways:
  - First the values can be obtained by hard coding them in the definition of constructor. In these cases the values to be assigned to variables will be fixed. Every time an object of a class is created it would be assigned those fixed values only.
  - Second approach is to let the constructor accept the arguments directly from the user. In such cases the constructor definition would typically contain Input statements that would accept values from the user and would use those values to assign member variables of the object of the class. This approach does not assign fixed values to each and every object that gets created and is more flexible compared to the previous approach.

Example of a Default constructor

```
class Student
{
	String name;
	String city;
	int age;
	int grade;
	Scanner sc = new Scanner(System.in);
	void display()
	{
		System.out.println(name +"\n"+city+"\n"+age+"\n"+grade);
	}
	//no argument constructor
	Student()  //name of the constructor has to be the same name as of its class
	{
		name = sc.next();
		city = sc.next();
		age = sc.nextInt();
		grade = sc.nextInt();
	}
};
```

Example of a Parameterized constructor

```
class Circle
{
        float r,area;
        Circle(float x)  //parameterized
constructor
        {
          r=x;
        }

        void calculate()
        {
          area=3.14f*r*r;
        }
        void display()
        {
          System.out.println("Area="+area);
        }
        Circle()
        {
                r=1;
        }
};
```

Constructor overloading

- It is possible to have two constructors in the same class. We can have a default constructor that would get invoked if no arguments are received and another parameterized constructor that would get invoked if arguments are passed.
- It is a process of creating more than one constructor in a class that have different set of parameters.
- The compiler decides which constructor to call based on the parameters passed at the time of object creation.

```
class Circle
{
        float r,area;
        Circle(float x)  //parameterized constructor
        {
          r=x;
        }

        void calculate()
        {
          area=3.14f*r*r;
        }
        void display()
        {
          System.out.println("Area="+area);
        }
        Circle()
        {
                r=1;
        }
};
```

Copy constructor

- There is also another type of a constructor that accepts arguments. But unlike parameterized constructor this one does not receive mere values but whole object as an arguments.
- After the object is passed as an argument to the constructor, it uses the values inside that object and assigns them to its own member variables.
- Hence these types of constructors are called as copy constructors since it copies the values of the whole object into its own variables.

Example of a copy constructor

```
import java.util.*;
class Circle
{
private float r,area;
Circle()
{
  Scanner sc= new Scanner(System.in);
  System.out.print("Enter Radius:");
  r=sc.nextFloat();
}
Circle(Circle x)
{
  r=x.r;
}
void calculate()
{
  area=3.14f*r*r;
}
void display()
{
  System.out.println("Area="+area);
}
}
```

```
class Main
{
public static void main(String args[])
{
  Circle c=new Circle();
  c.calculate();
  c.display();
  Circle c1=new Circle(c);
  c1.calculate();
  c1.display();
}
}
```

Explanation

- As you can see the Circle class has a constructor that accepts an object of class Circle as an argument. The constructor treats that object as x which has a variable to store radius r. It is accessed using x.r.
- When you see r = x.r; the r on the left is the r that belongs to the object of that constructor called. And x.r is the r of the object passed as an argument. Sometimes the same statement can also be written as this.r = x.r; the execution of both the statements gives the same result.
- The object in this case has just one variable, but it can have more than one variables that we can use

Another example of a Copy constructor, it accepts object with 2 variables.

```
import java.util.*;
class Euclid
{
private int n1,n2,gcd;
Euclid(Euclid x)
{
  n1=x.n1;
  n2=x.n2;
}
void calculate()
{
  int temp;
  while(n1%n2!=0)
 {
    n1=n1%n2;
    temp=n1;
    n1=n2;
    n2=temp;
  }
  gcd=n2;
}
void display()
{
  System.out.println("GCD="+gcd);
}
}
class Main
{
public static void main(String args[])
{
  Euclid e=new Euclid();
  e.calculate();
  e.display();
  Euclid e1=new Euclid (e);
  e1.calculate();
  e1.display();
}
}
```

Explanation

```
Euclid(Euclid x)
{
  n1=x.n1;
  n2=x.n2;
}
```

This is a copy constructor that receives an object as
An argument which contains two member variables
n1 and n2 which is used by this constructor to assign those v
to its own member variables.

This keyword

- This refers to current object.
- When executing statements in a method() and you wish to refer to the same object through which the method is called, then you can use “this” keyword to access the object

Code example:-

```
import java.util.*;
import java.io.*;
class Compare {
  int x;
  Compare(int a){
    x=a;
  }
  Compare compare(Compare c){
    if(x > c.x)
      return this;
    else
      return c;
  }
  void display(){
    System.out.println(x);
  }
}
class test{
  public static void main(String[] args) {
    Compare a = new Compare(10);
    Compare b = new Compare(20);
    a.display();
    b.display();
    Compare result = a.compare(b); //this -> a
    result.display();
    Compare result = b.compare(a); //this -> b
    result.display();

  }
```

Explanation:-

The class Compare has one variable and 2 methods.
It also has a constructor to initialize the members of a class
The compare method accepts an object as an argument and
Also returns an object.
This method is called by object a and object b is passed
as an argument.
If variable x of object b is greater than variable x of a,
then object b is returned.
The returned object is assigned to another object result.
On other hand if x of a is greater then that of b then object a
Is returned and assigned to result.
So when result.display() method is called,
it displays the x of the object that was returned.

Method overriding

```
import java.util.*;
import java.io.*;
class A{
  int a= 10;
   void read(){
    System.out.println("read base");
   }
}
class B extends A{
  int a =20;
  void read()
   {
    System.out.println("read derived");
   }
}
class test{
  public static void main(String[] args) {
    B b = new B();
    b.read();  // read of derived class overrides the
read of base
  }
}
```

When the method of a base class
Has same name and same arguments
Of a derived class, the method of
The derived class is always called.
This is called method overriding.

Static keyword – static class members

- When we create method within a class, we can usually access it through the instance of that class.
- Such methods are called "instance methods".
- However when we declare a method inside a class to be "static". We need not create an instance of that class to access such methods.
- They can be accessed through the name of the class as such "classname.methodname()".
- These methods work on the class level and not on the instance level.
- Similarly if a variable in a class i.e a field member is declared static it has similar properties.
- Only one copy of that variable gets created which is shared by all the instances created of that class.
- Static field members can be sued to keep a count of number of instances created for that class. When such a member is created, its single copy is shared by all instances. So each time a new instance gets created, a constructor of that class can increment that variable and it will be common to all instances created.