Constructors in Subclasses

constructor of sub class is invoked when we create the object of subclass, it by default invokes the default constructor of super class **(that has no parameters)**. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the **super keyword**, but it should be first statement in a constructor. The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is **not permitted**. The super keyword should be used when we need to call a constructor **with parameters**.

1. **Constructors are not inherited**. That is, if you extend an existing class to make a subclass, the constructors in the superclass do not become a part of the subclass.

2. If you want constructors in the subclass, you have to define new ones.

3. If you don't define any constructors in the subclass, then the computer will make up a default constructor, with no parameters, for you. This could be a problem, if there is a constructor in the superclass that does a lot of necessary work. It looks like you might have to repeat all that work in the subclass!
4. The constructor of the super class could be called by involves the special variable, **super**.

5. At the beginning statement in a constructor, we should use **super** to call a constructor from the superclass. The notation for this is a bit ugly and misleading, and it can only be used in this one particular circumstance: It looks like you are calling **super** as a method (even though **super** is not a method and you can't call constructors the same way you call other methods anyway).

Example 1 (Trace)

1- Case 1

```
package javaapplication55;
public class One {
   public One(){
      System.out.println("One");}
   public One(int a){
      System.out.println("One with parameter");}
   }

package javaapplication55;
public class Two extends One{
   public Two(){
      super(); // optional
      System.out.println("Two");}
public Two(int a){
     super(); //optional
   System.out.println("Two parameter"); }  }
```

The output will be:

```
One
Two
One
Two parameter
```

2- Case 2

```
package javaapplication55;
public class Two extends One{
   public Two(){
      super(4);
      System.out.println("Two");}
public Two(int a){
     super(5);       System.out.println("Two parameter");}  }
```

The output will be:

```
One with parameter
Two
One with parameter
Two parameter
```

3- Case 3

```
package javaapplication55;
public class One {
   public One(){
      System.out.println("One");}
   public One(int a){
      System.out.println("One with parameter");}
   }
public class Two extends One{
   public Two(){
    System.out.println("Two");}

public Two(int a){

   System.out.println("Two parameter");
} }

public class JavaApplication55 {
public static void main(String[] args) {

      Two t1=new Two();
      Two t2=new Two(6);

}}
```

The output will be:

```
Two
Two parameter
```

Example2 (trace)

```
public class One  {
    protected int x,y;
    public One(int a,int b){
        x=a;y=b;
    System.out.println("One"+x+"   "+y);}
    public One() {
        System.out.println("One One");}    }

public class Two extends One{
   public Two(){
            System.out.println("Two");} }

public class Main  {
   public static void main(String[] args){
   Two t=new Two ( );  } }
```

The output will be :

```
One One
Two
```

Case 1:

If we delete the empty constructor of the One class ….the compiler will detect an error …….why?

Case 2:

If we delete both constructors what will happen? Why?

Case3:

If we delete the One(int , int) constructor what will be the output ? why?

Case4:

If we call the super constructor as shown:

```
public class Two extends One {
   public Two(){
            super (3,4);
            System.out.println("Two");}}
```

The output will be:

```
One 3   4
Two
```

The statement “**super** (3,4);” calls the constructor from the superclass. **<u>This call must be the first line of the constructor in the subclass</u>**. Note that if you don’t explicitly (واضح) call a constructor from the superclass in this way, then the constructor from the superclass, the one with no parameters, will be called automatically. If there is no constructor with no parameters … the compiler will detect error. **<u>Repeat the same cases which discussed previously but with calling super(3,4)…and compare the results of these three cases.</u>**

Example2 (Writing a program)

Emergency contacts

Crisis alert systems are all the rage these days. When an emergency manifests itself, all folks who have registered with the emergency contact database are notified via email, phone, text message , etc. We can use the concepts of inheritance to reduce the system's complexity and allow for future ways of contacting individuals.

First, we model the general Contact. All contacts should have a name, but the particular way in which they are contacted depends upon their preferred method of communication. We will leave it for the subclasses of Contact to decide how to implement the notify method.

```
public class Contact {
  private String firstName;
  private String lastName;

  public Contact(String givenFirstName,
   String givenLastName) {
    firstName = givenFirstName;
    lastName = givenLastName;   }

  public String getName() {
    return (firstName + " " + lastName);   }}
```

Defining a subclass using the extends keyword

Now, let's make an EmailContact which is a subclass of Contact that is specialized for email notification. In order to define a subclass of any other class, we have to use an extends clause:

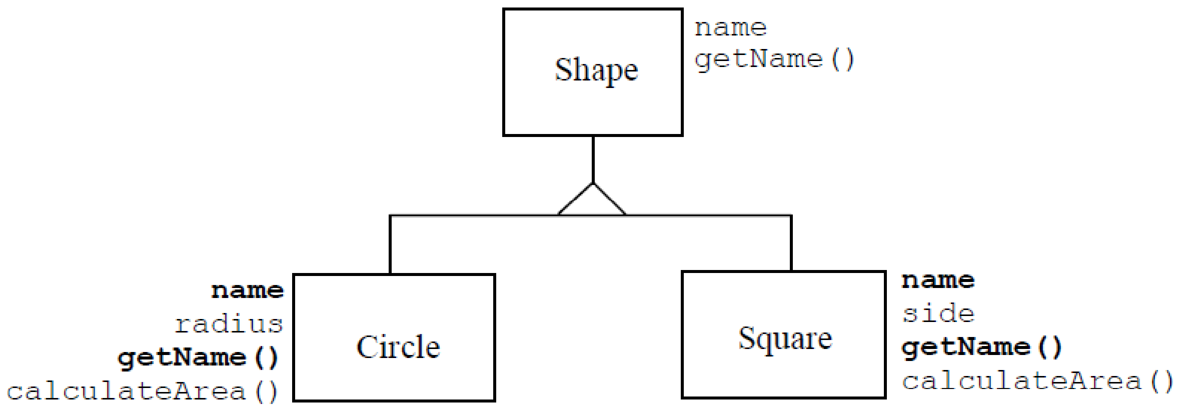
```
    public class EmailContact extends Contact {
      private String emailAddress;

public EmailContact(String givenFirstName, String givenLastName,
                          String givenEmailAddress)   {
        // first, we call the superclass constructor to initialize the
        // "inherited" instance variables
        super(givenFirstName, givenLastName);

        // then, initialize everything that is special for EmailContact
        emailAddress = givenEmailAddress;
      }

      public void notify(String alertMessage)
      {
        // send an email to the address
        System.out.println("Esteemed " + getName() + ",");
        System.out.println(alertMessage);
      }
    }
public class EmergencyTester {
public static void main(String[] args) {
EmailContact ec=new EmailContact("Yasser","Mohammed","iraq@gmail.com");
ec.notify("FIRE near School HIGH");

      }
    }
```

The output will be:

```
Esteemed Yasser Mohammed,
FIRE near School HIGH
```

Example LAB :

The following figure is a class hierarchy of shapes. Shape is a generalized class of Circle and Square. All shapes have a name and a measurement by which the area of the shape is calculated. The attribute name and method getName() are defined as properties of Shape. Circle and Square, being subclasses of Shape, inherit these properties (highlighted in bold in the following figure).use constructors to set the values of attributes.



```
public class Shape {
private String name;
public Shape(String aName) {name=aName;}
public String getName() {return name;}  }

public class Circle extends Shape{
private int redius;
Circle(String aName) {
super(aName);
radius = 3; }
public double calculateArea() {
  return (3.14 * radius * radius);
class Square extends Shape {
private int side;
Square(String aName) {
super(aName);
side = 3; }
public double calculateArea() {
return (side*side);
} }
```

Complete the program and execute it in the Lab