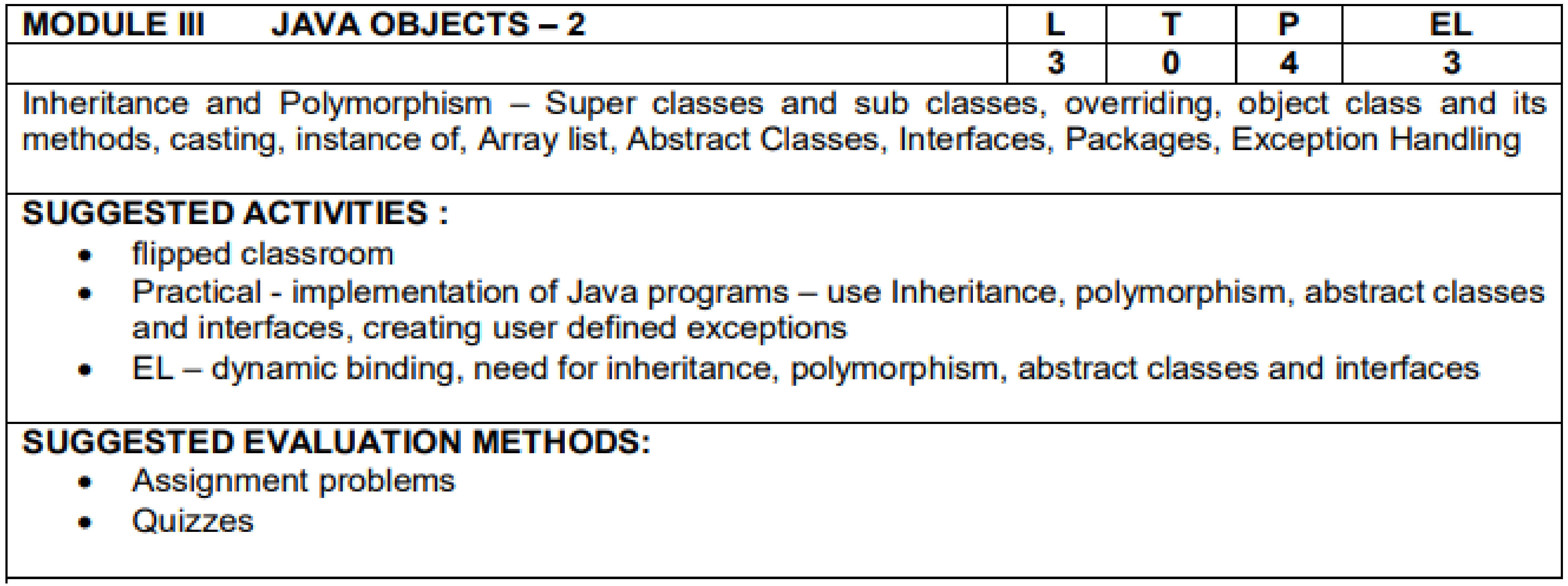
CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Syllabus



MODULE III JAVA OBJECTS – 2	**L**	**T**	**P**	**EL**
	3	**0**	**4**	**3**
Inheritance and Polymorphism – Super classes and sub classes, overriding, object class and its methods, casting, instance of, Array list, Abstract Classes, Interfaces, Packages, Exception Handling				

SUGGESTED ACTIVITIES :

- flipped classroom
- Practical - implementation of Java programs – use Inheritance, polymorphism, abstract classes and interfaces, creating user defined exceptions
- EL – dynamic binding, need for inheritance, polymorphism, abstract classes and interfaces

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Quizzes

Object class methods and its purpose

Object clone()

Creates a new object that is the same as the object being cloned.

boolean equals(Object)

Compares two Objects for equality.

Void finalize()

Called by the garbage collector on an object to reclaim the memory of the unused object.

Class<?>getClass()

Returns the class of an object at run time.

int hashCode()

Returns a hash code value associated with the invoking object.

notify()

Resumes the execution of a single thread that is waiting on the invoking object.

notifyAll()

Resumes the execution of all threads that is waiting on the invoking object.

toString()

Returns a string that describes the object.

wait()
wait(long)
wait(long, int)

Waits to be notified by another thread of execution.

Clone() method of Object class

- protected Object clone() throws CloneNotSupportedException
- Object class is the super class of all the classes.
- Clone method of Object class creates an new copy of the object exactly same as the referred object by declaring that the class implements Cloneable interface otherwise throws CloneNotSupportedException
- Clone method can create shallow copy and Deep copy.
    - Returns: a clone of the object.
    - Throws: CloneNotSupportedException
        - In case if the object's class does not implement the Cloneable interface then the subclasses that override the clone method can throw this exception to indicate that an instance cannot be cloned.
    - Throws: OutOfMemoryError
        - if there is not enough memory.

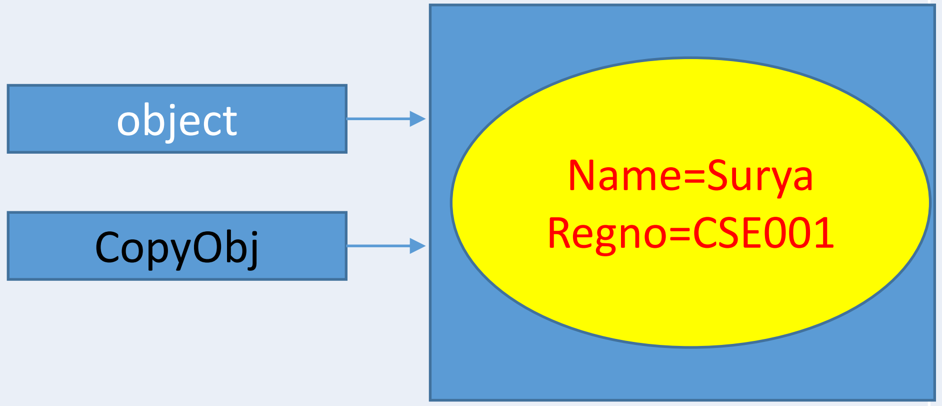
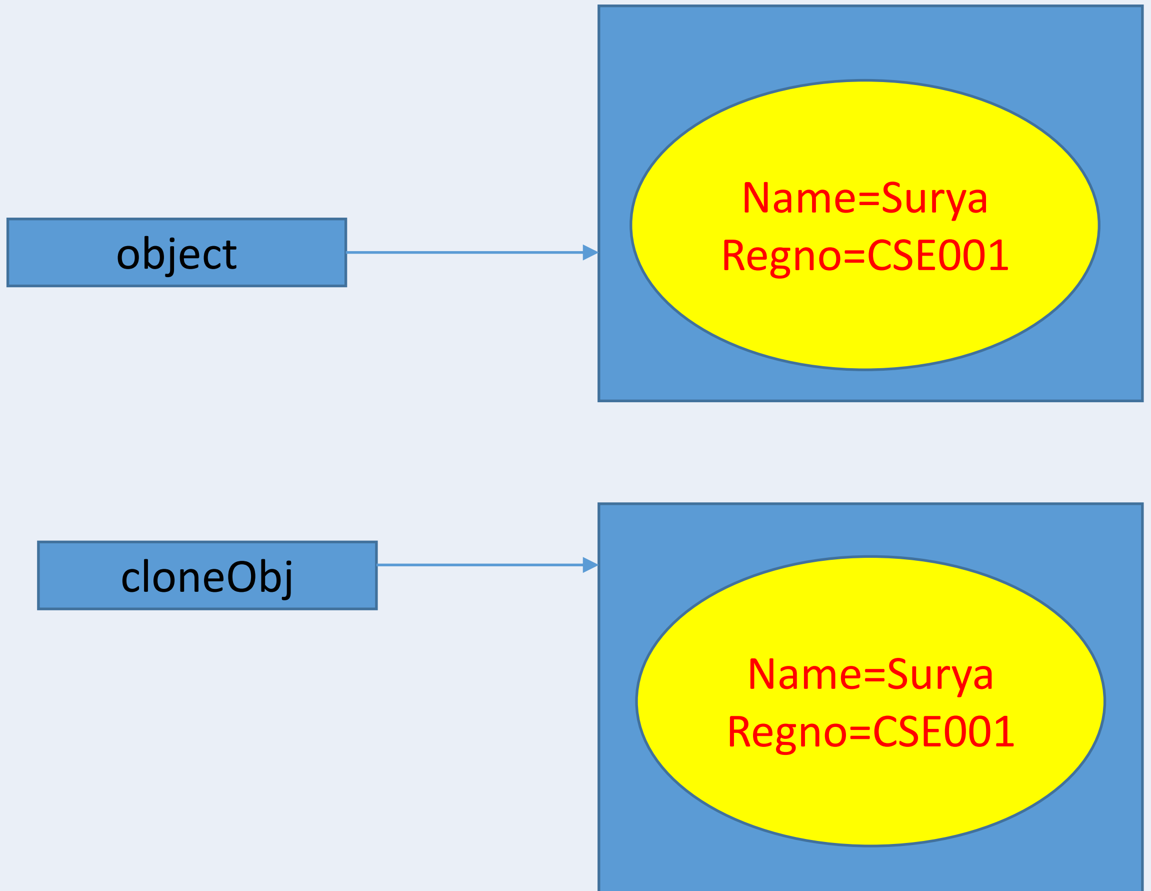
Object class and its method

- public interface Cloneable
    - A class implements the Cloneable interface to indicate to the clone method can make a field-for-field copy of instances of that class.
    - Attempts to clone instances that do not implement the Cloneable interface will throw an exception CloneNotSupportedException.

Assignment copy vs clone method copy

Assignment copy	**Clone() copy**

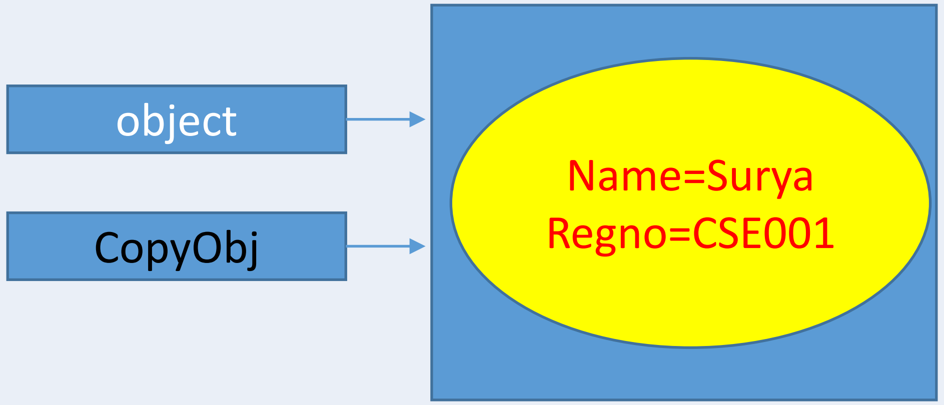
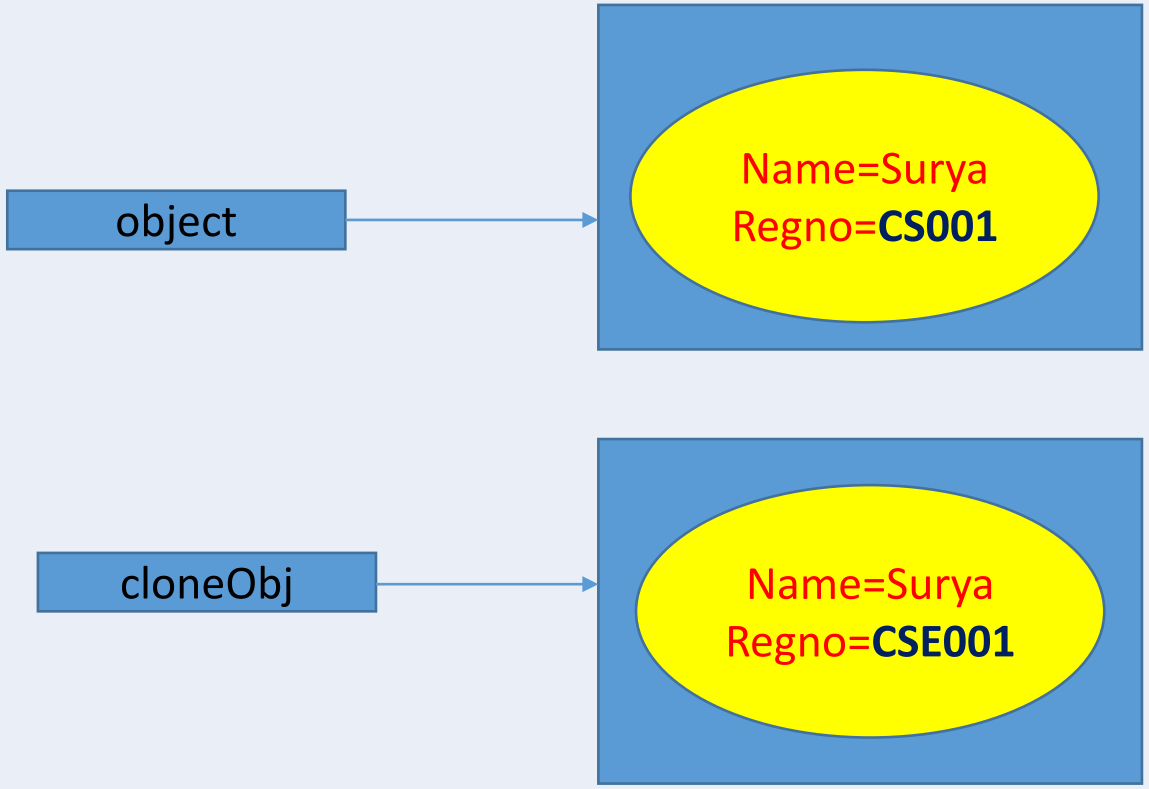
Student object = **new** Student();
object.Name=Surya
object.Regno=CSE001
Student CopyObj = object;
//**object** and CopyObj point to the same location
	Student object = **new** Student();
	object.Name=Surya
	object.Regno=CSE001
	Student cloneObj = (Student) object.clone();
	//object and clone object remain independent

	 object
	Name=Surya
	Regno=CSE001
	cloneObj
	Name=Surya
	Regno=CSE001

Assignment copy vs clone method copy

Assignment copy	

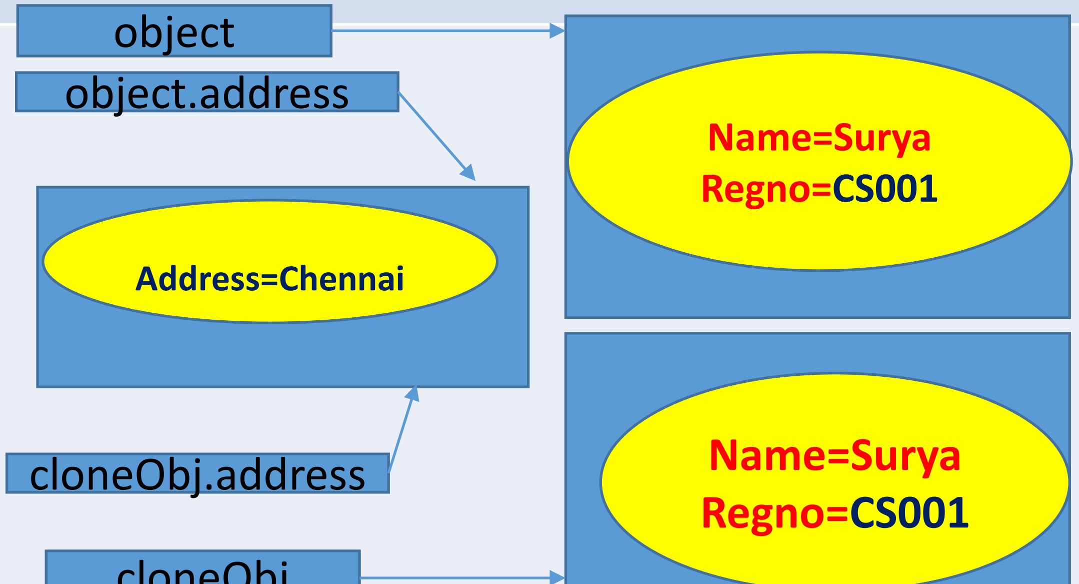
Student object = **new** Student();
object.Name=Surya;
object.Regno=CSE001;
Student CopyObj = object;
object.Regno=CS001;
//any changes in **object** will be reflected in **CopyObj**
	Student object = **new** Student();
	object.Name=Surya;
	object.Regno=CSE001;
	Student cloneObj = (Student) object.clone();
	object.Regno=CS001;
	//any changes in **object** will not be reflected in **cloneObj**.

	 object
	Name=Surya
	Regno=CS001
	cloneObj
	Name=Surya
	Regno=CSE001

Shallow copy vs Deep copy

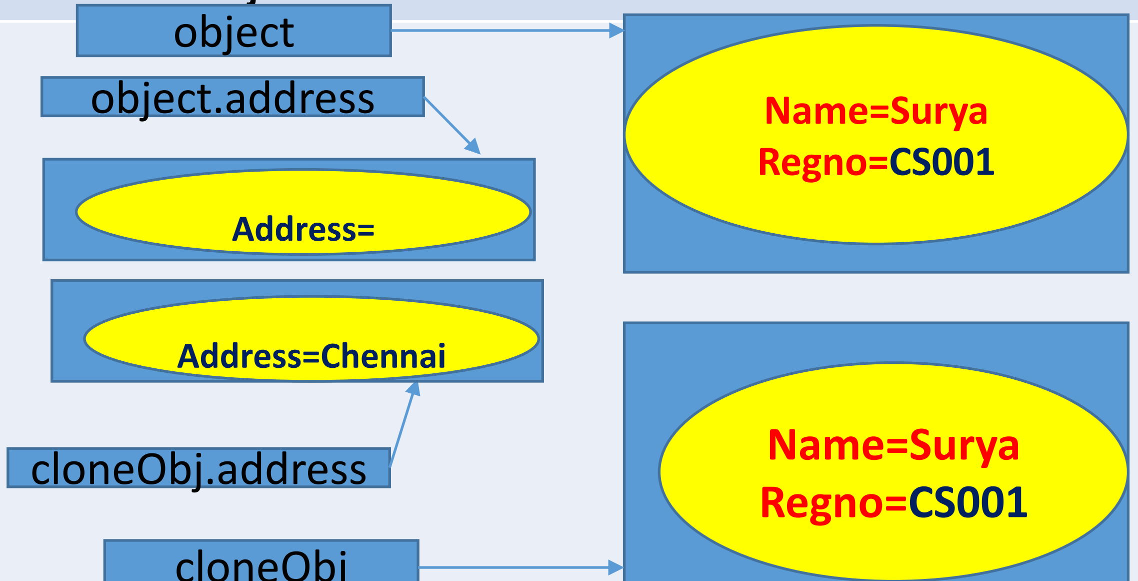
```
Student object = new Student();
object.Name=Surya;
object.Regno=CSE001;
Student cloneObj = (Student) object.clone();
object.Regno=CS001;
Address address=new Address();
object.address="Chennai";
```

//copying from **one object** to another object(only Primitive type data members). Any changes made in the referenced object member **object** or **cloneObj** will be reflected in the other.



```
Student object = new Student();
object.Name=Surya;
object.Regno=CSE001;
object.Regno=CS001;
Address address=new Address();
object.address="Chennai";
Student cloneObj = (Student) object.clone();
cloneObj.address = (Address) object.address.clone();
object.address="Delhi";
```

//copying everything from **one object** to another object(Primitive and class type data members). Any changes made in the referenced object member in **object** or **cloneObj** will not be reflected in the other.



Assignment copy

```
class Student implements Cloneable {
int Regno;
String Name;
public Object clone() throws CloneNotSupportedException{
return super.clone(); }
}
public class Main
{
public static void main(String[] args) throws CloneNotSupportedException  {
Student object = new Student();
object.Name = “Surya";
object.Regno = CSE001;
Student cloneObj = (Student) object.clone();
object.Regno = CS001;
System.out.println(“object member details:");
System.out.println("Name:"+object.Name + " Regno:"+ object.Regno);
System.out.println(" Clone object member details :");
System.out.println("Name:"+ cloneObj.Name + " RollNo:"+ cloneObj.Regno);
System.out.println(“Equality of Object and clone object:" + (object == cloneObj));
}
}
```

cloning

```
class Student {
int Regno;
String Name;
}
public class Main
{
public static void main(String[] args) throws CloneNotSupportedException  {
Student object = new Student();
object.Name = “Surya";
object.Regno = CSE001;
Student cloneObj = object;
object.Regno = CS001;
System.out.println(“object member details:");
System.out.println("Name:"+object.Name + " Regno:"+ object.Regno);
System.out.println(" Clone object member details :");
System.out.println("Name:"+ cloneObj.Name + " RollNo:"+ cloneObj.Regno);
System.out.println(“Equality of Object and clone object:" + (object == cloneObj));
}
}
```

Shallow cloning

```
class Student implements Cloneable {
int Regno;
String Name;
Address address = new Address();
public Object clone() throws CloneNotSupportedException{
return super.clone(); }
}
public class Main
{
public static void main(String[] args) throws CloneNotSupportedException  {
Student object = new Student();
object.Name = “Surya";
object.Regno = CSE001;
object.address = new Address(600042, "Chennai");
Student cloneObj = (Student) object.clone();
System.out.println(“object member details:");
System.out.println("Name:"+object.Name + " Regno:"+ object.Regno);
System.out.println("Address:"+ object.address.pincode + object.address.city);
System.out.println(" Clone object member details :");
System.out.println("Name:"+ cloneObj.Name + " RollNo:"+ cloneObj.Regno);
System.out.println("Address:"+ cloneObj.address.pincode + " and " + cloneObj.address.city);
System.out.println(“Equality of Object and clone object:" + (object == cloneObj));
System.out.println(" Equality of address Object and cloned address object:" :"+(object.address == cloneObj.address));
}
}
```

```
class Address implements Cloneable{
int pincode;
String city;
Address(){ }
Address(int pincode, String city){
 this.pincode = pincode;
 this.city = city; }
public Object clone() throws
CloneNotSupportedException {
return super.clone();  }
}
```

Deep cloning

```
class Student implements Cloneable {
int Regno;
String Name;
Address address = new Address();
public Object clone() throws CloneNotSupportedException{
return super.clone(); }
}
public class Main
{
public static void main(String[] args) throws CloneNotSupportedException  {
Student object = new Student();
object.Name = “Surya";
object.Regno = CSE001;
object.address = new Address(600042, "Chennai");
Student cloneObj = (Student) object.clone();
cloneObj.address = (Address) object.address.clone();
System.out.println(“object member details:");
System.out.println("Name:"+object.Name + " Regno:"+ object.Regno);
System.out.println("Address:"+ object.address.pincode + object.address.city);
System.out.println(" Clone object member details :");
System.out.println("Name:"+ cloneObj.Name + " RollNo:"+ cloneObj.Regno);
System.out.println("Address:"+ cloneObj.address.pincode + " and " + cloneObj.address.city);
System.out.println(“Equality of Object and clone object:" + (object == cloneObj));
System.out.println(" Equality of address Object and cloned address object:" :"+(object.address == cloneObj.address));
}
}
```

```
class Address implements Cloneable{
int pincode;
String city;
Address(){ }
Address(int pincode, String city){
 this.pincode = pincode;
 this.city = city; }
public Object clone() throws
CloneNotSupportedException {
return super.clone();  }
}
```

hashCode

- hashCode invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer.
- This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the equals method, then calling the hashCode method on each of the two objects must produce the same integer result.
- This method is supported for the benefit of hashtables.

equals()

- public boolean equals(Object obj)
- This method returns true if and only if x and y refer to the same object.
- For any reference value x, x.equals(x) returns true
- For any reference value x and y, x.equals(y) returns true, if y.equals(x) returns true.
- Parameters:
  - obj - the reference object with which to compare.
- Returns:
  - true if this object is the same as the obj argument; false otherwise

toString()

- public String toString()
- Returns a string representation of the object
- Subclasses override this method.
- The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object.

- Returns:
    - a string representation of the object.

finalize()

- protected void finalize() throws Throwable
- Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.
- Throws: Throwable

```
public class Example
{
public static void main(String[] args)
{
String str = "Hai";
str = null;
System.gc();
System.out.println("Garbage collector");
}

protected void finalize()
{
System.out.println(“finalize executed");
}
}
```

Output: Garbage collector

```
public class Example
{
public static void main(String[] args)
{
Example obj = new Example();
obj = null;
System.gc();
System.out.println("Garbage collector");
}
protected void finalize()
{
System.out.println(" finalize executed ");
}
}
```

Output: Garbage collector
finalize executed

```
public class Example
{
public static void main(String[] args)
{
Example obj = new Example();
Example obj1 = new Example();
obj1 = null;
obj.finalize(); //explicit invoke
System.out.println("Garbage collector");
System.gc(); //implicit invoke
}
protected void finalize()
{
System.out.println(" finalize executed ");
}
}
```

Output: finalize executed
Garbage collector
finalize executed

```
public class Example
{
public static void main(String[] args)
{
Example obj = new Example();
Example obj1 = new Example();
obj1 = null;
obj.finalize(); //explicit invoke
System.out.println("Garbage collector");
obj.callMe(); //will throw IOException as invoked by destroyed object
System.gc();
}
protected void finalize()
{
System.out.println(" finalize executed ");
}
Public void callMe()
{
System.out.println(“I will not be executed”);
}
```

JAVA Collections:ArrayList

- The ArrayList class implements the List interface.
- ArrayList is a generic class that has this declaration:
  - class ArrayList<E>
  - Here, E specifies the type of objects that the list will hold.
- ArrayList supports dynamic arrays that can grow as needed.
- In Java, standard arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that how many elements an array will hold must be known in advance.
- But, sometimes, until run time precisely how large an array is needed is not known.
- To handle this situation, the Collections Framework defines ArrayList.

- **Create an ArrayList**

```
ArrayList<String> list=new ArrayList<String>();
ArrayList<Integer> list=new ArrayList<Integer>();
```

- **add elements to an ArrayList**
  - **Use add() method**
  - **list.add(“Steve Jobs"); //This will add "Steve Jobs" at the end of List**
  - **list.add(2, "Steve Jobs"); //This will add "Steve Jobs" at the third position**

```
import java.util.*;
class JavaExample{
  public static void main(String args[]){
    ArrayList<String> list=new ArrayList<String>();
    list.add("Steve Jobs");
    list.add("Tim Cook");

    list.add(1, " Ali baba "); //Adding “Ali baba" at the 1st position

    System.out.println(list); //displaying elements

     list.set(0, “Charless Babage"); // Change an element in ArrayList
     String s=list.get(0);
     System.out.println(list); //displaying elements

     list.remove("Steve Jobs"); //to remove elements from ArrayList

     list.remove(0) ; //to remove elements from ArrayList
     //iterating ArrayList
             for(String str:list)
                System.out.println(str);
      System.out.println(list.size()); // ArrayList Size
   }    }
```

add(Object o): adds an object o to the arraylist.

add(int index, Object o): adds the object o to the array list at the given index.
remove(Object o): Removes the object o from the ArrayList.
remove(int index): Removes element from a given index.
set(int index, Object o): Used for updating an element. It replaces the element present at the specified index with the object o.
int indexOf(Object o): Gives the index of the object o. If the element is not found in the list then this method returns the value -1.
Object get(int index): It returns the object of list which is present at the specified index.
int size(): It gives the size of the ArrayList – Number of elements of the list.
boolean contains(Object o): It checks whether the given object o is present in the array list if its there then it returns true else it returns false.
clear(): It is used for removing all the elements of the array list in one go.
obj.clear();

```
import java.util.ArrayList;
import java.util.Collections;
class JavaExample{
   public static void main(String args[]){
      ArrayList<String> list=new ArrayList<String>();
      list.add("Steve Jobs");
      list.add("Tim Cook");
        Collections.sort(list);
     //iterating ArrayList
                  for(String str:list)
                       System.out.println(str);
         System.out.println(list.size()); // ArrayList Size
        int pos = list.indexOf("Tim Cook ");
}    }
```