```
toString() -> return "";
boolen equals(Object obj){
if(obj instanceof Date)
Date d = (Date) obj;
this.day == d.day
}
```

d1==d2

```
class Date{

}

Date d1 = new Date(1,1,2000);
Date d2 = new Date(1,1,2000);

d1.equals(d2);
```

100% incomplete -> abstract
abstract class

Super

1,2,3,4,5,........50

Interface - Java 7

interface
It provides set of rules

```
interface Acceptable{
void accept(Scanner sc);
}
```

```
interface Displayable{
void display();
}
```

```
class Employee implements Acceptable{
void accept(Scanner sc){

}
}
```

```
class Time implements Acceptable{
void accept(Scanner sc){

}
}
```

```
Acceptable a = new Employee(); // upcasting
a.accept();
Acceptable a = new Time(); // upcasting
a.accept();
```

1,2,3,4,5......50

Collection Framework
Collection -> Interface
List
Set
Map
Queue

```
class Manager{

}
```

```
class Salesman{

}
```

void* -> int*

```
class SalesManager : Manager, Salesman{

}
```

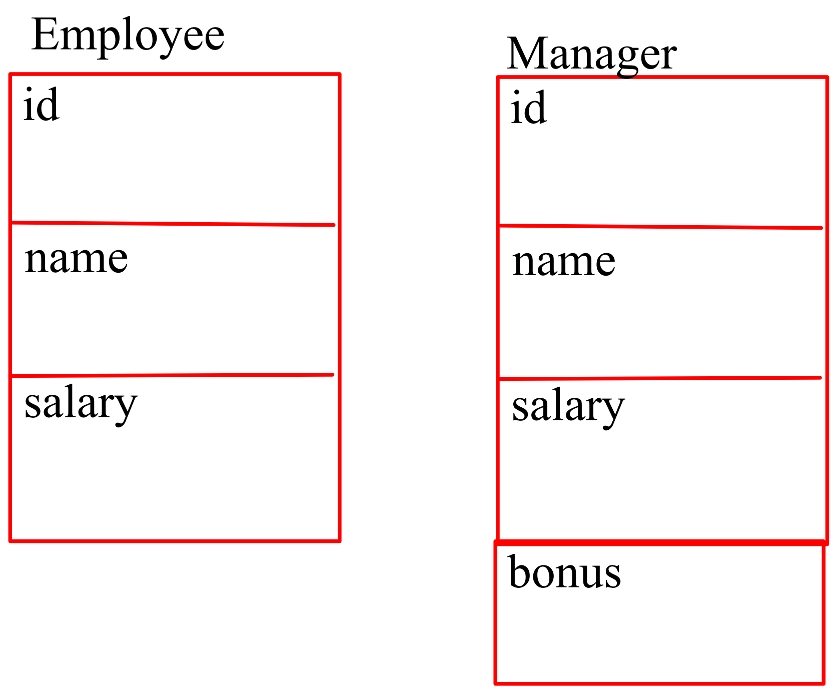
```
Manager *m = new SalesManager();// upcasting
//Salesman *s = new SalesManager();//upcasting
Salesman *s = (Salesman) m;
```

class
- We can instantiate a class
- It consists of static as well as non static fields and methods
- we can declare a constructor
- It can have only non abstract methods

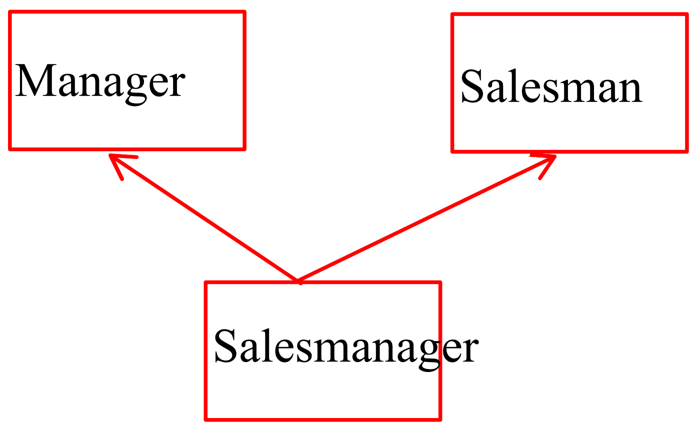
abstract class
- We cannot instantiate an abstract class
- It consists of static as well as non static fields and methods
- we can declare a constructor
- It can have abstract as well as non abstract methods

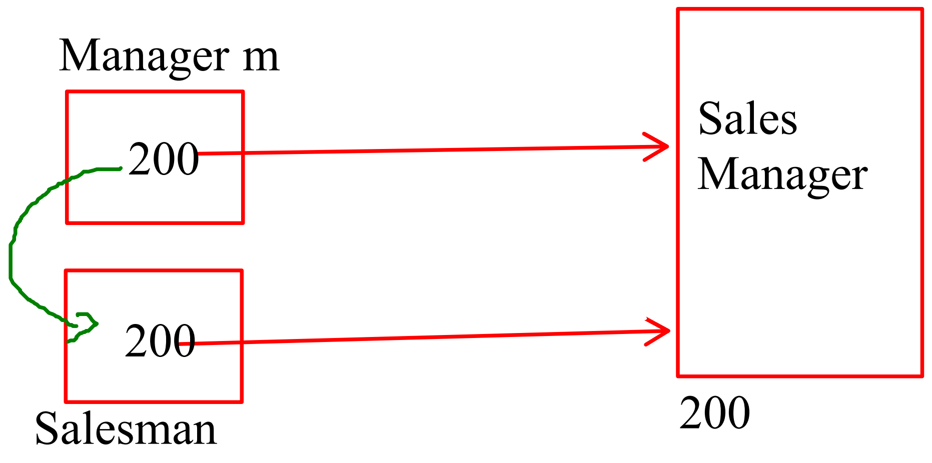
interface
- We cannot instantiate an interface
- It consists of public static final fields and abstract methods
- we cannot declare a constructor
- It by default have only abstract methods.

```
interface Acceptable{
int n1=10;
int n2=20;
void accept(Scanner sc);
}

abstract class Employee implements Acceptable{
id,
name,
salary
}

class Manager extends Employee {
bonus;

void accept(Scanner sc){

}
}
```



new Manager();

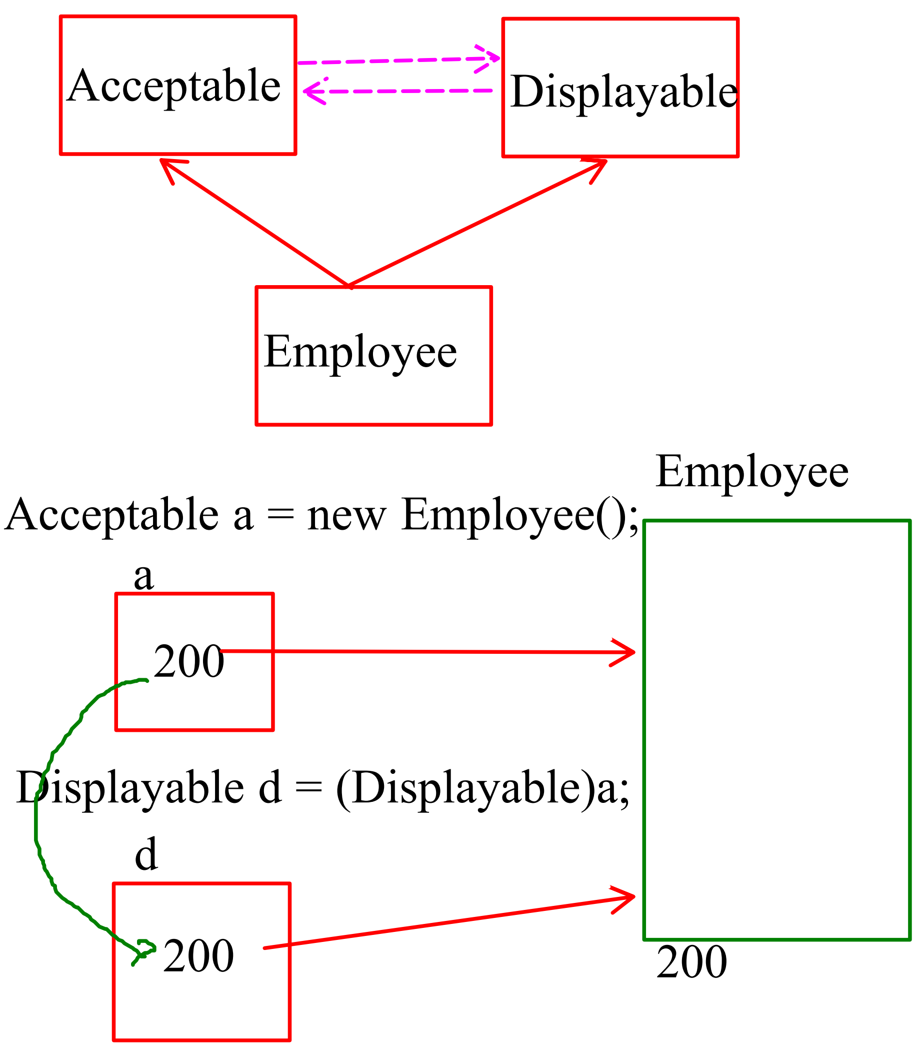




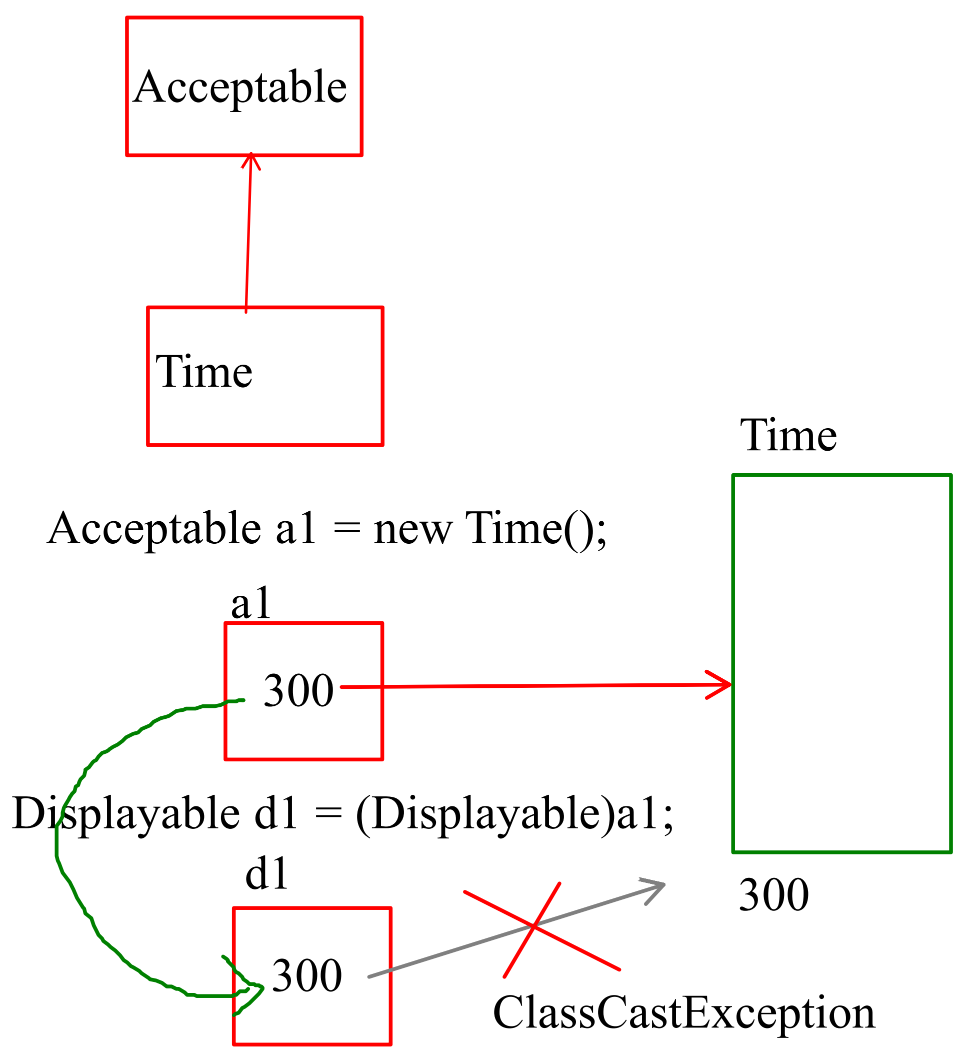
Manager m = new Salesmanager(); // upcasting

//Salesman s = new Salesmanager(); // upcasting

Salesman s = (Salesman) m;



```
Acceptable a = new Employee();
Displayable d = (Displayable)a;
```



```
Acceptable a1 = new Time();
Displayable d1 = (Displayable)a1;
```

Marker Interface
- An empty interface is called as a marker interface
- Marker interface is also called as the tagging interface
- It is used to provide the extra information/metadata to the JVM
- eg -> Cloneable, Seralizable

```
// Marker Interface
interface I1{
}
```

```
 class Fruit{
        String color;
        double weight;
        String name;
        boolean isFresh = true;

        Fruit(String name){
                this.name = name;
        }
        Fruit(String name, weight, color){
                this.name = name;
        }
        String toString(){
                return color+","+weight+","+name;
        }

        public String taste(){
                return "no specific taste";
        }
 }


     int counter = 0;

     size = sc.nextInt();
     Fruit [] basket = new Fruit[size];
```

```
class Mango extends Fruit{
        Mango(){
                super("Mango");
        }
        Mango(String nm, weight, color){
                super(nm,weight,color)
        }
}

 class Apple extends Fruit{
        Apple(){
                super("Apple");
        }
 }


    if(counter<size)
    basket[counter] = new Mango("mango",sc.nextDou
    counter++;
    basket[1] = new Apple("apple");


  for(Fruit f : basket)
  if(f !=null && f.getIsFresh()){
        sysout(f)
        sysout(f.taste())
  }
```

```
for(Fruit f:basket)
{
        String t = f.taste();
        if(t.equals("sour"))
                f.setIsFresh(false);
}
```

clone
String

JVM archictecture
Buzzwords
GC

## Exception Handling

1. Errors
2. Exceptions

$$\begin{tikzcd} & \text{Throwable} & \\ \text{Error} \arrow[ur] & & \text{Exception} \arrow[ul] \end{tikzcd}$$

1. try
2. catch
3. throw
4. throws
5. finally

```
try(create the resources that have implemented AutoClosable interface){
// to check for the exceptions
}
catch(Exception e){
// handle the exception
}
finally{
// to close the resources
sysout("Inside Finally");
}
```

## Exception
1. Checked Exception
    - Exception class and its subclasses except RunTimeException class are all considered as Checked Exception
    - It is compulsary to handle Checked Exception else compiler generates an error
2. Unchecked Exception
    - RuntimeException class and its subclasses are considered as unchecked Exception
    - It is not mandatory to handle the unchecked Exceptions

# Lab
1. Complete the assignment
2. DO the classwork (Interface,Exceptions)
2.1 Custom Exception
3. access modifiers
4. Rules of method overriding
5. upcasting and downcasting
6. Shallow copy & Deep copy