Object Oriented Programming with Java

Sandeep Kulange

Inheritance

- If "is-a" relationship is exist between the types then we should use inheritance.
- Inheritance is also called as generalization.
- Example
    1. Manager is a employee
    2. Book is a product
    3. Triangle is a shape
    4. SavingAccount is a account.

```
class Employee{ //Parent class
    //TODO
}
class Manager extends Employee{ //Child class
    //TODO
}
//Here class Manager is extended from class Employee.
```

Inheritance

- If we want to implement inheritance then we should use extends keyword.
- In Java, parent class is called as super class and child class is called as sub class.
- Java do not support private and protected mode of inheritance
- If Java, class can extend only one class. In other words, multiple class inheritance is not allowed.
- Consider following code:

```
class A{     }
class B{     }
class C extends A, B{    //Not OK
}
```

Inheritance

- During inheritance, if super type and sub type is class, then it is called as implementation inheritance.

Single implementation Inheritance

```
class A{     }
class B extends A{ }     //OK
```

Hierarchical implementation Inheritance

```
class A{     }
class B extends A{ } //OK
class C extends A{ } //OK
```

Multiple implementation Inheritance

```
class A{     }
class B{     }
class C extends A, B{ } //Not OK
```

Multilevel implementation inheritance

```
class A{     }
class B extends A{ } //OK
class C extends B{ } //OK
```

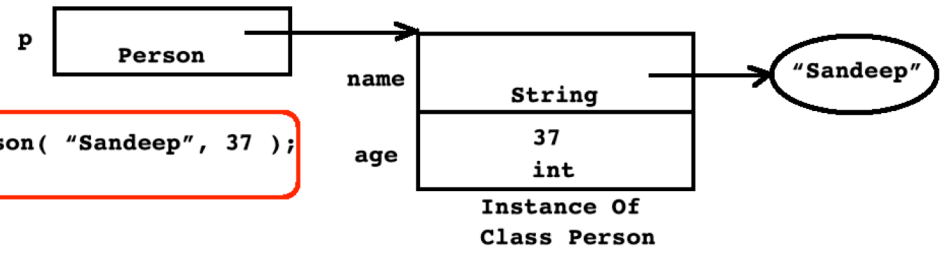
Multiple Inheritance In Java

```
class A{     }
class B{     }
class C extends A, B{    //Not OK : Multiple implementation inheritance
    //TODO
}
```

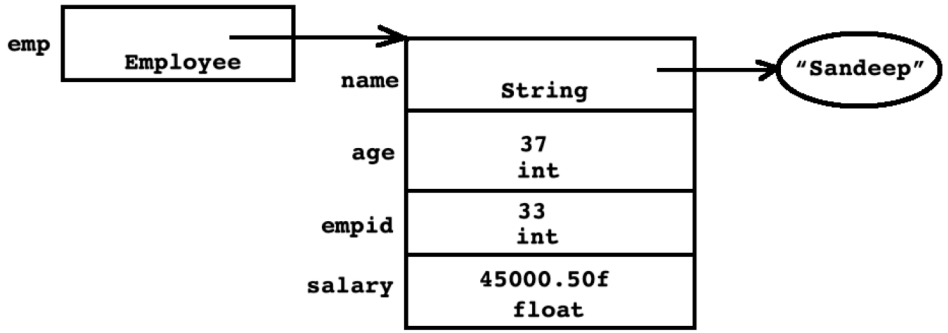
```
interface A{     }
interface B{     }
interface C extends A, B{    //OK : Multiple interface inheritance
    //TODO
}
```

```
interface A{     }
interface B{     }
class C implements A, B{    //OK : Multiple interface implementation inheritance
    //TODO
}
```

Inheritance In Java




```
Person p = new Person( "Sandeep", 37 );
p.showRecord( );
```




```
Employee emp = new Employee("Sandeep", 37,33,45000.50f);
emp.displayRecord( );
```

Inheritance

- If we create instance of sub class then all the non static fields declared in super class and sub class get space inside it. In other words, non static fields of super class inherit into sub class.
- Static field do not get space inside instance. It is designed to share among all the instances of same class.
- Using sub class, we can access static fields declared in super class. In other words, static fields of super class inherit into sub class.
- All the fields of super class inherit into sub class but only non static fields gets space inside instance of sub class.
- Fields of sub class, do not inherit into super class. Hence if we create instance of super class then only non static fields declared in super class get space inside it.
- If we declare field in super class static then, all the instances of super class and sub class share single copy of static field declared in super class.

Inheritance

- We can call/invoke, non static method of super class on instance of sub class. In other words, non static method inherit into sub class.

- We can call static method of super class on sub class. In other words, static method inherit into sub class.

- Except constructor, all the methods of super class inherit into sub class.

Inheritance

- If we create instance of super class then only super class constructor gets called. But if we create instance of sub class then JVM first give call to the super class constructor and then sub class constructor.
- From any constructor of sub class, by default, super class's parameterless constructor gets called.
- Using super statement, we can call any constructor of super class from constructor of sub class.
- Super statement, must be first statement inside constructor body.

Inheritance

- According to client's requirement, if implementation of super class is logically incomplete / partially complete then we should extend the class. In other words we should use inheritance.
- According to client's requirement, if implementation of super class method is logically incomplete / partially complete then we should redefine method inside sub class.
- Process of redefining method of super class inside sub class is called as method overriding.

Inheritance

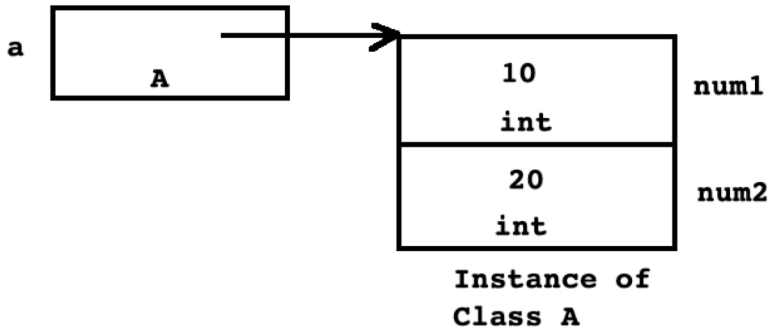
- If name of super class and sub class method is same and if we try to call such method on instance of sub class then preference is given to the method of sub class. This is called shadowing.
- Using super keyword, we can access members of super class inside method of sub class.
- During inheritance, members of sub class do not inherit into super class. Hence using super class instance, we can access members of super class only.
- During inheritance, members of super class, inherit into sub class. Hence using sub class instance, we can access members of super class as well as sub class.

Inheritance

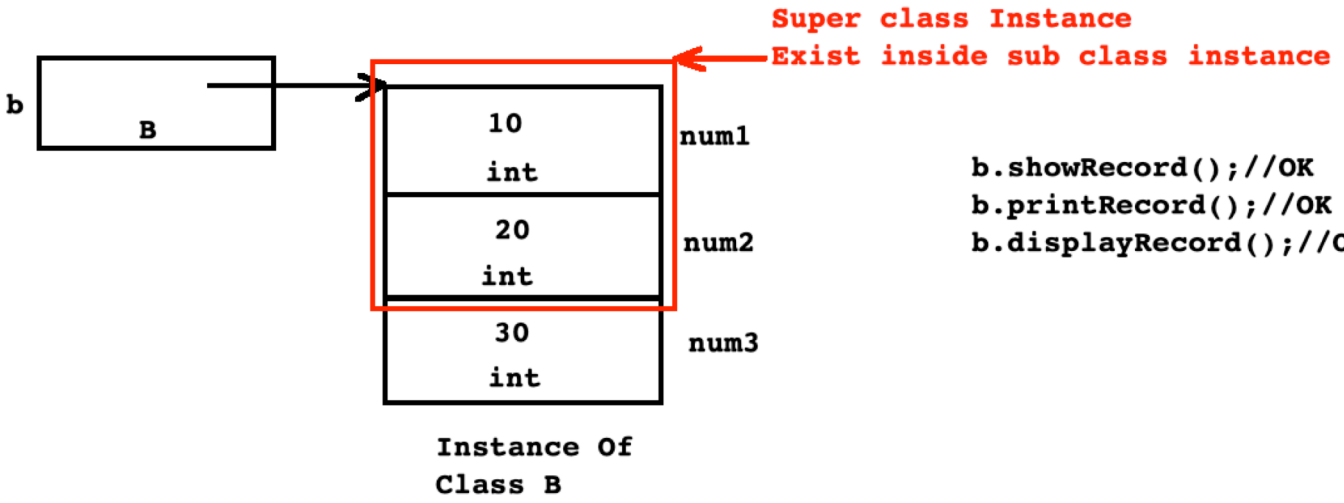
- Members of super class, inherit into sub class. Hence we can consider, sub class instance as a super class instance.
- Example : Employee is a Person
  - Since Employee contains all the properties and behavior of person hence Employee is a person.
- Since sub class instance can be considered as super class instance, we can use it in place super class instance.

```
Employee emp1 = null;    //OK

Employee emp2 = new Employee( );    //OK

person p = new Person( );
Employee emp3 = p;    //NOT OK

Employee emp4 = new Person( );  //NOT OK
```

Inheritance




```
a.showRecord();//OK
a.printRecord();//OK
a.displayRecord();//NOT OK
```




```
b.showRecord();//OK
b.printRecord();//OK
b.displayRecord();//OK
```

Upcasting

- Process of converting reference of sub class into reference of super class is called as upcasting.

```
Employee emp = null;
Person p = ( Person )emp;    //OK : Upcasting
//p : null
//emp : null
```

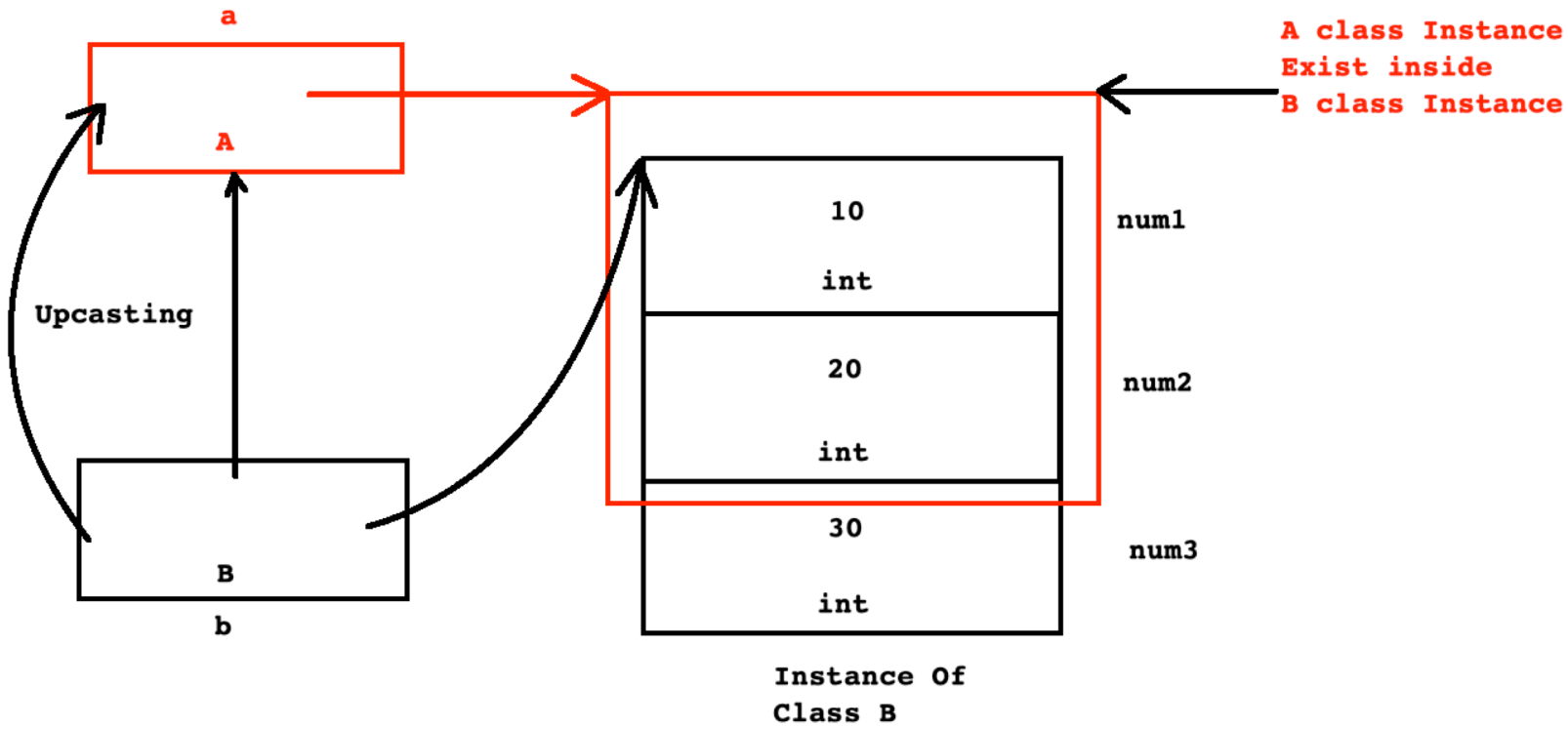
```
Employee emp = new Employee( );
//Person p = ( Person )emp;    //OK : Upcasting
Person p = emp;    //OK : Upcasting
```

- Using upcasting, we can minimize object dependency in the code. In other words, we can reduce maintenance of the code.

- During upcasting, explicit type casting is optional.

- Super class reference variable can contain reference of sub class instance. It is also called as upcasting.
  - **Person p = new Employee( ); //Upcasting**

Upcasting




```
B b = new B( );
A a = ( A )b; //Upcasting

B b = new B( );
A a = b;  //OK : Upcasting
```

```
A a = new B( ); //OK : Upcasting
```

Upcasting

- In case of upcasting, using super class reference variable, we can access:
  1. Fields of super class
  2. Methods of super class
  3. overridden methods of sub class

- In case of upcasting, using super class reference variable, we can not access:
  1. fields of sub class
  2. Non overridden methods of sub class.

- If we want to access fields and non overridden methods of sub class then we should do down casting.

Down Casting

- Process of converting reference of super class into reference of sub class is called down casting.

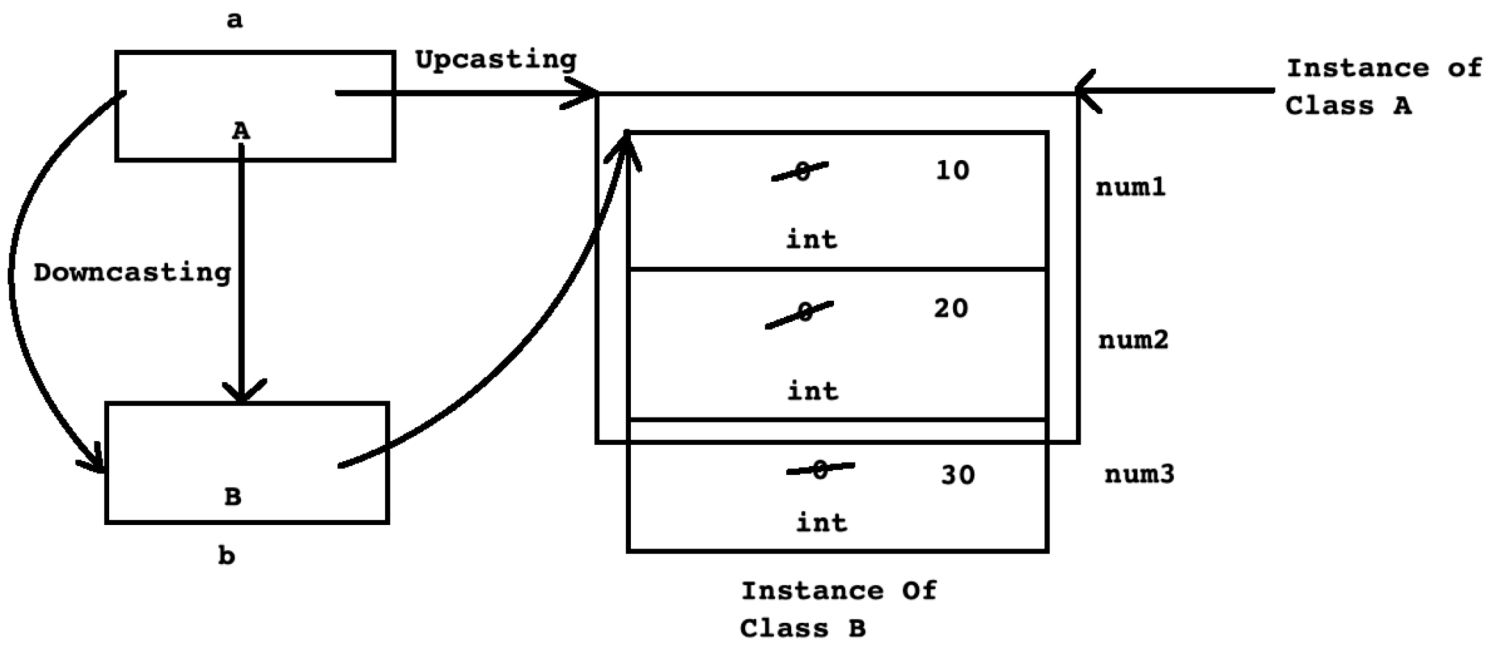
```
Person p = new Employee( ); //Upcasting
Employee emp = ( Employee)p;    //Downcasting : OK
```

```
Person p = null;
Employee emp = ( Employee)p; //OK : Downcasting
//p = null
//emp = null
```

```
Person p = new Person();
Employee emp = ( Employee)p; //Downcasting : ClassCastException
```

- Only in case of upcasting, we should do down casting. Otherwise JVM will throw ClassCastException.
- In case of upcasting, using super class reference variable, we can access overridden method of sub class. It is also called as dynamic method dispatch.

Down Casting



Method Overriding

- Process of redefining method of super class inside sub class is called method overriding.
- Method redefined in sub class is called overrided method.
- When we call method of sub class using reference of super class then it is called dynamic method dispatch.

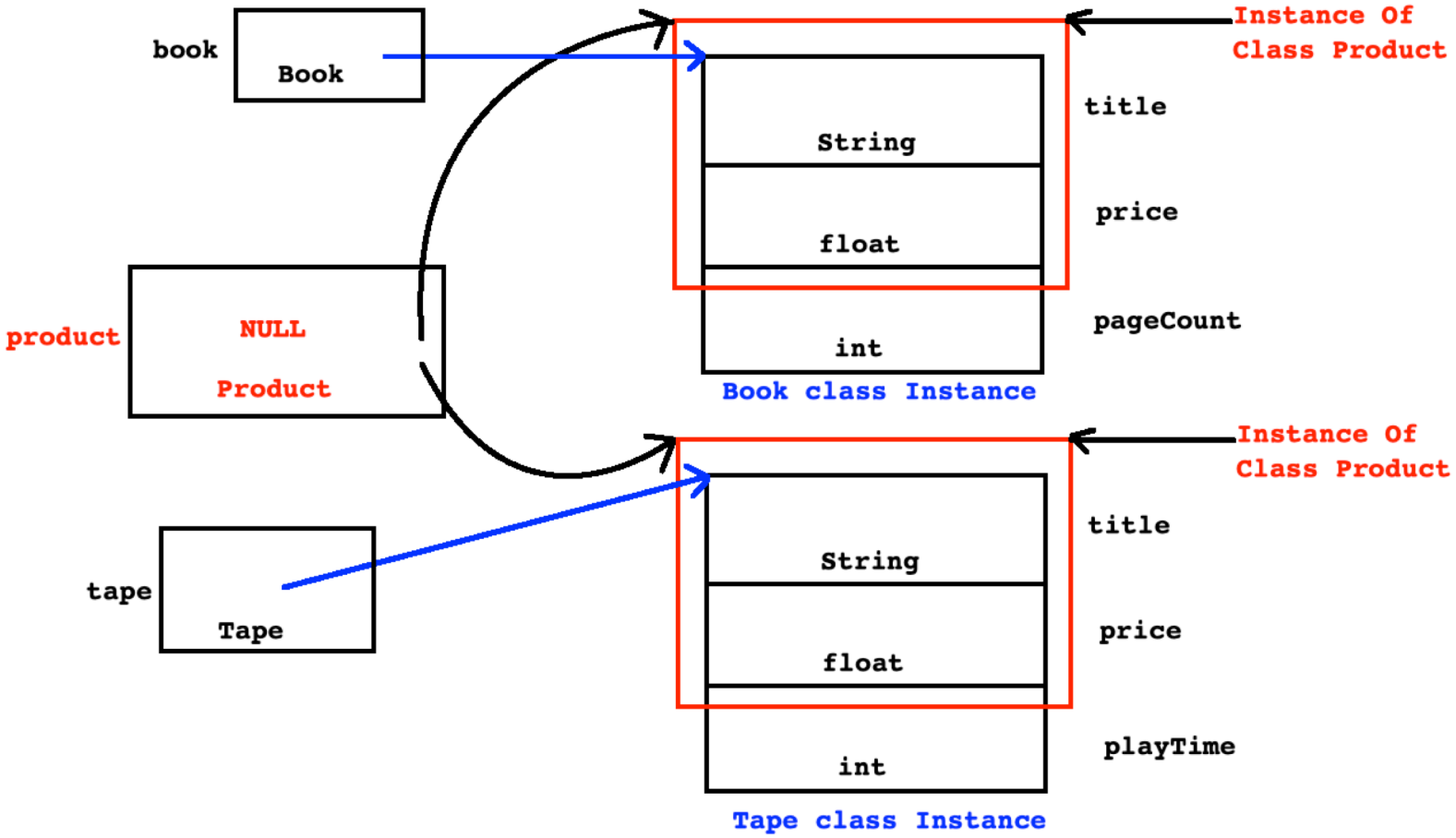
```
class A{
    public void print( ) {
        System.out.println("A.print");
    }
}
class B extends A{
    public void print( ) {
        System.out.println("B.print");
    }
}
public class Program {
    public static void main(String[] args) {
        A a = new B( ); //Upcasting
        a.print();  //Dynamic Method Dispatch
    }
}
```

Method Overriding

- If implementation of super class method is partially complete then we should redefine/override method inside sub class.
- **Rules of method overriding**
    1. Access modifier in sub class method should be same or it should be wider.
    2. Return type of sub class method should be same or it should be sub type. In other words, it should be covariant.
    3. Name of the method, number of parameters and type of parameters inside sub class method must be same.
    4. Checked exception list inside sub class method should be same or it should be sub set.

- Override is annotation declared in java.lang package.
- We can use this annotation on method only.
- It helps developer to override method inside sub class.

Down Casting



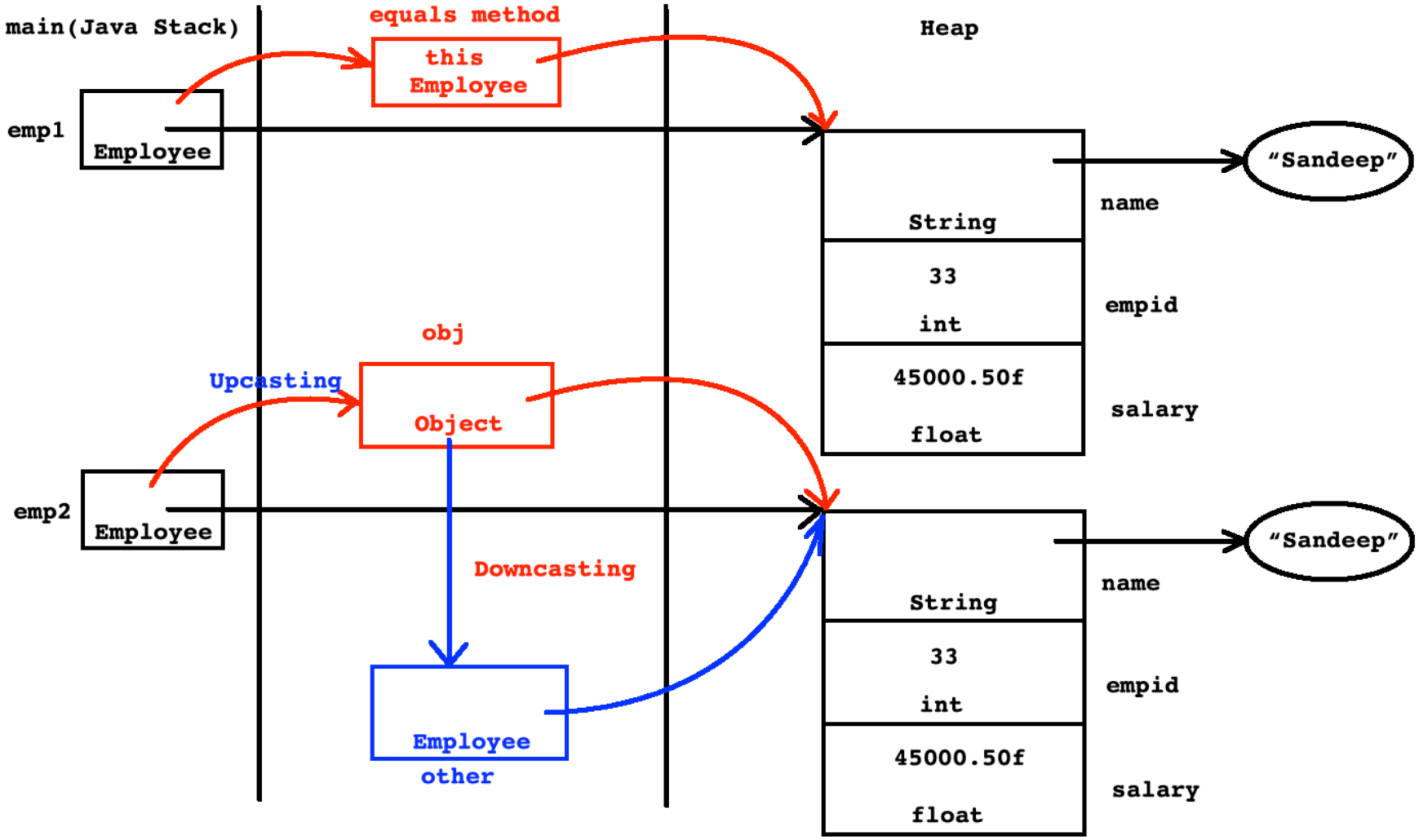
equals( ) Method

- equals is non final method of java.lang.Object class.
- Syntax:
  - **public boolean equals(Object obj);**
- Consider its implementation inside Object class.

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- If we do not define equals method inside class then super class's equals method gets called.
- equals method of Object class do not compare state of instances. It compares state of references.
- If we want to compare state of instances then we should override equals method inside class.

equals( ) Method



equals( ) Method

```
// Employee this = emp1;
// Object obj = emp2; //Upcasting
@Override
public boolean equals(Object obj) {
    if (obj != null) {
        Employee other = (Employee) obj;
        if (this.empid == other.empid)
            return true;
    }
    return false;
}
```

Final Method

- If implementation of super class method is logically 100% complete then we should declare super class method final.
- We can not redefine/override final method inside sub class.
- Since final method inherit into sub class, we can use it inside sub class.
- Example:
    1. name(), ordinal( )
    2. 6 methods of Object class : getClass(), wait(), notify(), notifyAll() etc.
- We can declare overridden method final.
- In java we can not override following methods:
    1. Static Method
    2. Final Method
    3. Private method

Abstract Method

- abstract is keyword in Java.
- If implementation of super class method is logically 100% incomplete then we should declare super class method abstract.
- Abstract methods do not specify a body.
- If we declare method abstract then it is mandatory to declare class abstract.
- It is mandatory to override abstract method of super class otherwise sub class will be considered as abstract.

```
abstract class A{
    public abstract void f1( );
}
```

```
class B extends A{
    @Override
    public void f1( ){
    }
}
```

```
abstract class B extends A{
}
```

Abstract Class

- We can not instantiate abstract class. But we can create reference of abstract class.
- Example:
    1. java.lang.Number
    2. java.lang.Enum
    3. java.util.Calendar
    4. java.util.Dictionary
    5. java.io.InputStream
    6. java.io.OutputStream
    7. java.io.Reader
    8. java.io.Writer
- Without declaring method abstract, we can declare class abstract. In other words, abstract class may/may not contain abstract method.

Final Class

- If implementation of any class is logically 100% complete then we should declare such class final.
- We can not extend final class. In other words, we can not create sub class of final class.
- Example:
    1. java.lang.System
    2. java.lang.String/StringBuffere/StringBuilder
    3. All Wrapper Classes
    4. java.lang.Math
    5. java.util.Scanner

Sole Constructor

- A constructor of super class, which is designed to call from constructor of sub class is called sole constructor.

```
abstract class A{
    private int num1;
    private int num2;
    public A(int num1, int num2) {  //Sole Constructor
        this.num1 = num1;
        this.num2 = num2;
    }
}
class B extends A{
    private int num3;
    public B(int num1, int num2, int num3) {
        super(num1, num2);
        this.num3 = num3;
    }
}
public class Program {
    public static void main(String[] args) {
        A a = new B( 100, 200, 300 );
        a.print();
    }
}
```

Thank You.

[sandeepkulange@sunbeaminfo.com]