Object Oriented Programming with Java 8
PG-DAC

Rohan Paramane

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

- 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.

Regarding this and super

this(...) implies invoking constructor from the same class.
super(...) implies invoking constructor from the immediate super class

1. Only a constr can use this() or super()
2. Has to be 1st statement in the constructor
3. Any constructor can never have both ie. this() & super()
4. super & this (w/o brackets) are used to access (visible) members of super class or the same class.

Upcasting

When the reference variable of super class refers to the object of subclass, it is known as widening or **upcasting in java**.

when subclass object type is converted into superclass type, it is called widening or upcasting.

Superclass s = new SubClass();

Up casting : *Assigning child class object to parent class reference .*
Syntax for up casting : **Parent p = new Child();**
Here **p** is a parent class reference but point to the child object. *This reference p can access all the methods and variables of parent class but only overridden methods in child class.*

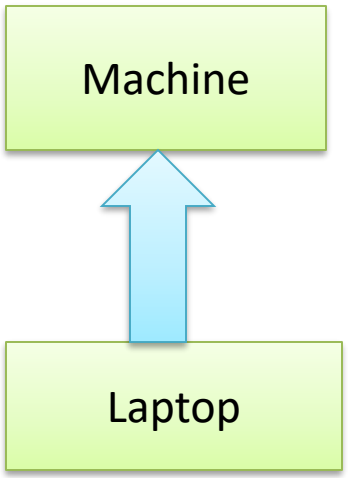
Upcasting gives us the flexibility to access the parent class members, but it is not possible to access all the child class members using this feature. Instead of all the members, we can access some specified members of the child class. For instance, we can only access the overridden methods in the child class.

Downcasting

Down casting : *Assigning parent class reference (which is pointing to child class object) to child class reference .*

Syntax for down casting : **Child c = (Child)p;**

Here **p** is pointing to the object of child class as we saw earlier and now we cast this parent reference **p** to child class reference **c**. *Now this child class reference **c** can access all the methods and variables of child class as well as parent class.*



*For example, if we have two classes, **Machine** and **Laptop** which extends **Machine** class. Now for upcasting, every laptop will be a machine but for downcasting, every machine may not be a laptop because there may be some machines which can be **Printer**, **Mobile,** etc.*
Downcasting is not always safe, and we explicitly write the class names before doing downcasting. So that it won't give an error at compile time but it may throw **ClassCastExcpetion** at run time, if the parent class reference is not pointing to the appropriate child class.

Explanation

```
Machine machine = new Machine ();

Laptop laptop = (Laptop)machine;//this won't give an error while compiling
```

//laptop is a reference of type Laptop and machine is a reference of type Machine and points to Machine class Object .So logically assigning machine to laptop is invalid because these two classes have different object strucure.And hence throws ClassCastException at run time .

To remove ClassCastException we can use instanceof operator to check right type of class reference in case of down casting .

```
if(machine instanceof Laptop)
{
Laptop laptop = machine;      //here machine must be pointing to Laptop class object .
}
```

Polymorphisim

- The ability to have many different forms
- An object always has only one form
- A reference variable can refer to objects of different forms

Method Binding

Static Binding

- Static binding
- Compile time
- early binding
- Resolved by java compiler
- Achieved via method overloading

Example :
In class Test :

```
void test(int i,int j){...}
void test(int i) {..}
void test(double i){..}
void test(int i,double j,boolean flag){..}
int test(int a,int b){...}   //javac error
```

Dynamic Binding

- Dynamic binding
- Run time / Late binding
- Resolved by java runtime environment
- Achieved by method overriding (Dynamic method dispatch)
- Method Overriding is a Means of achieving run-time polymorphism

All java methods can be overridden : if they are not marked as private or static or final
Super-class form of method is called as overridden method
sub-class form of method is called as overriding form of the method
Example :

```
class A {
   A getInstance()
            {
              return new A();
            }
   }
```

```
class B extends A
{
   B getInstance()
            {
              return new B();
            }
}
```

Run time polymorphism or Dynamic method dispatch

- Super -class ref. can directly refer to sub-class object(direct=w/o type casting) as its the example of up-casting(similar to widening auto. conversion) .

- When such a super class ref is used to invoke the overriding method then the method to send for execution that decision is taken by JRE & not by compiler.
- In such case overriding form of the method(sub-class version) will be dispatched for exec.
- Javac resolves the method binding by the type of the reference & JVM resolves the method binding by type of the object it's referring to.
- Super class reference can directly refer to sub-class instance BUT it can only access the members declared in super-class directly.
- eg : A ref=new B();
  ref.show(); // this will invoke the sub-class: overriding form of the show () method

Java compiler resolves method binding by type of the reference & JVM resolves it by the type of the object, reference is referring to.

Interface

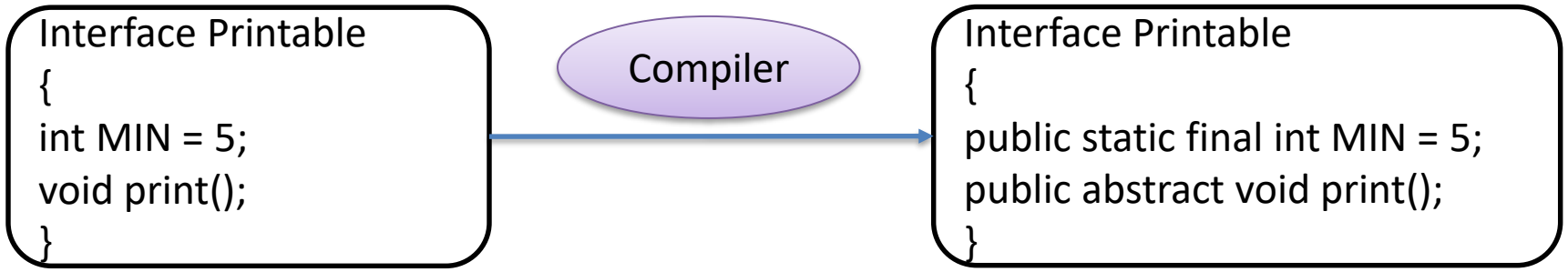
- In Java, an **interface** is a blueprint or template of a class. It is much similar to the Java class but the only difference is that it has abstract methods and static constants.
- An interface provides specifications of what a class should do or not and how it should do. An interface in Java basically has a set of methods that class may or may not apply.
- It also has capabilities to perform a function. The methods in interfaces do not contain any body.
- An interface in Java is a mechanism which we mainly use to achieve abstraction and multiple inheritances in Java.
- An interface provides a set of specifications that other classes must implement.
- We can implement multiple Java Interfaces by a Java class. All methods of an interface are implicitly public and abstract. The word abstract means these methods have no method body, only method signature.
- Java Interface also represents the IS-A relationship of inheritance between two classes.
- An interface can inherit or extend multiple interfaces.
- We can implement more than one interface in our class.

Interface Vs Class

- Unlike a class, you cannot instantiate or create an object of an interface.
- All the methods in an interface should be declared as abstract.
- An interface does not contain any constructors, but a class can.
- An interface cannot contain instance fields. It can only contain the fields that are declared as both static and final.
- An interface can not be extended or inherited by a class; it is implemented by a class.
- An interface cannot implement any class or another interface.

Syntax Interface

```
interface interface-name
{
//abstract methods
}
```

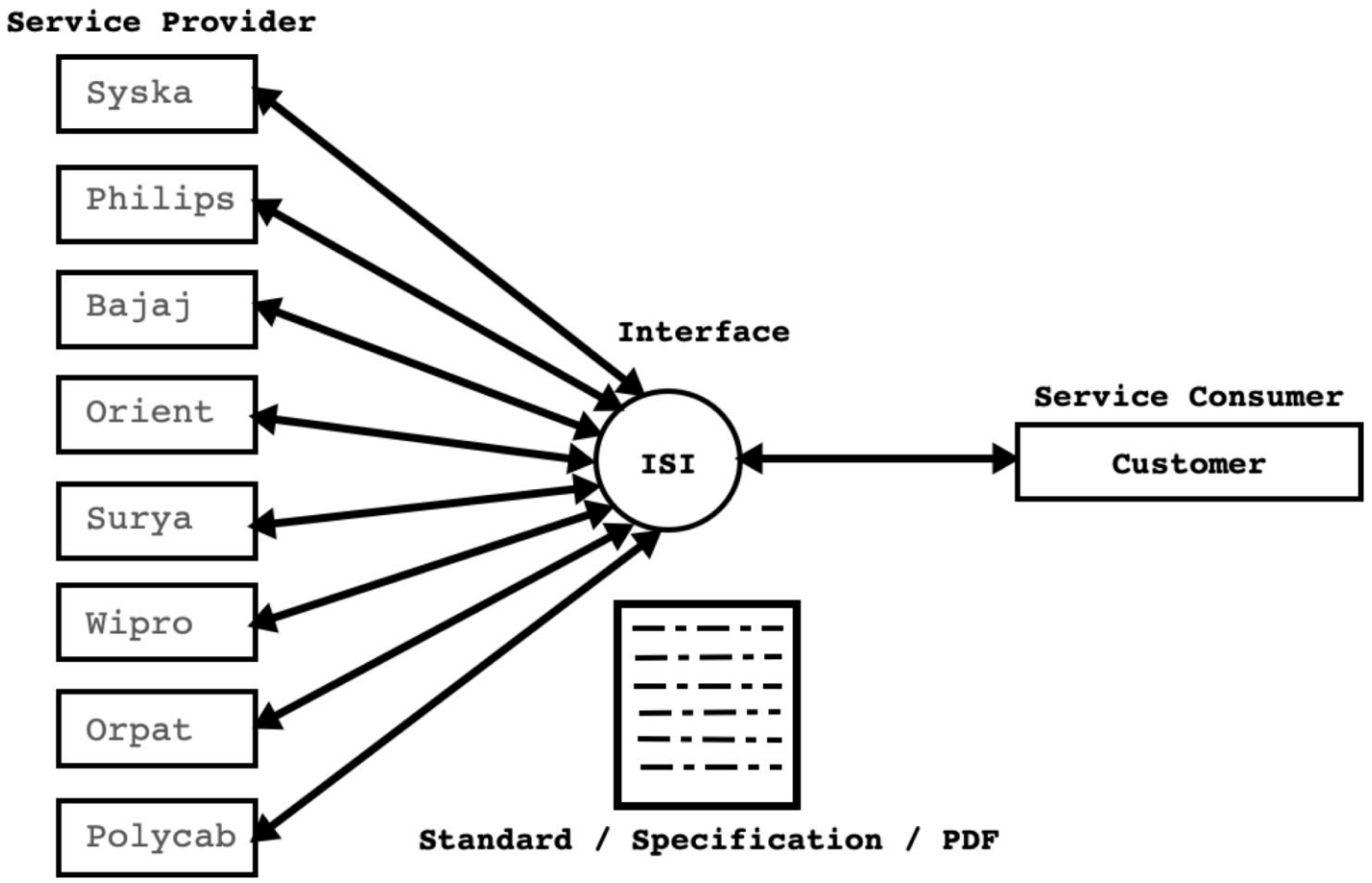


Interface

- Set of rules are called specification/standard.
- It is a contract between service consumer and service provider.
- If we want to define specification for the sub classes then we should define interface.
- Interface is non primitive type which helps developer:
  1. To build/develop trust between service provider and service consumer.
  2. To minimize vendor dependency.
- interface is a keyword in Java.

```
interface Printable{
    //TODO
}
```

Interface


Service Provider
Syska
Philips
Bajaj
Orient
Surya
Wipro
Orpat
Polycab
Interface
ISI
Service Consumer
Customer
Standard / Specification / PDF

Interface Syntax

Interface : I1, I2, I3
Class : C1, C2, C3

* I2 implements I1 //Incorrect
* I2 extends I1 //correct : Interface inheritance
* I3 extends I1, I2 //correct : Multiple interface inheritance
* C2 implements C1 //Incorrect
* C2 extends C1 //correct : Implementation Inheritance
* C3 extends C1,C2 //Incorrect : Multiple Implementation Inheritance
* I1 extends C1 //Incorrect
* I1 implements C1 //Incorrect
* c1 implements I1 //correct : Interface implementation inheritance
* c1 implements I1,I2 //correct : Multiple Interface implementation inheritance
* c2 implements I1,I2 extends C1 //Incorrect
* c2 extends C1 implements I1,I2 //correct

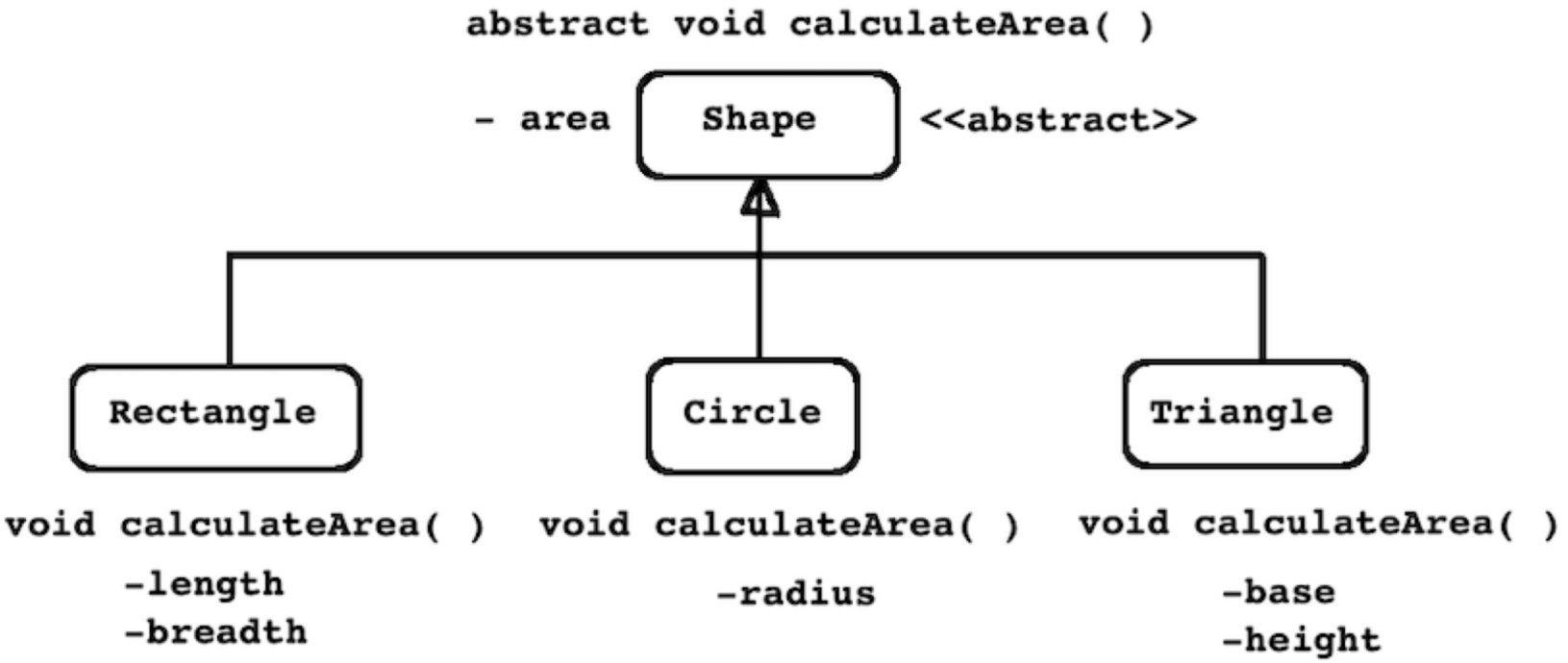
Types of inheritance

- **Interface Inheritance**
  - During inheritance if super type and sub type is interface then it is called interface inheritance.

1. Single Inheritance(Valid in Java)
2. Multiple Inheritance(Valid in Java)
3. Hierarchical Inheritance(Valid in Java)
4. Multilevel Inheritance(Valid in Java)

- **Implementation Inheritance**
  - During inheritance if super type and sub type is class then it is called implementation inheritance.
    1. Single Inheritance(Valid in Java)
    2. Multiple Inheritance(Invalid in Java)
    3. Hierarchical Inheritance(Valid in Java)
    4. Multilevel Inheritance(Valid in Java)

Abstract Class



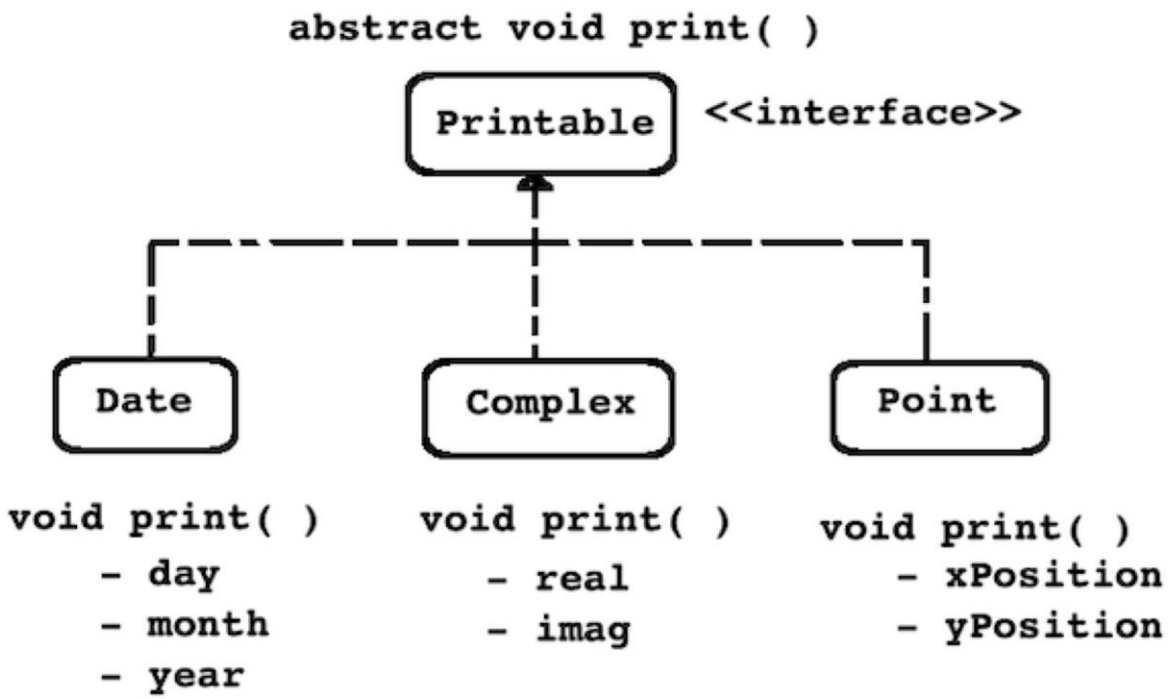

```
Shape[] arr = new Shape[ 3 ];
arr[ 0 ] = new Rectangle( ); //Upcasting
arr[ 1 ] = new Circle( ); //Upcasting
arr[ 2 ] = new Triangle( ); //Upcasting
```

Abstract Class

1. If "is-a" relationship is exist between super type and sub type and if we want same method design in all the sub types then super type must be abstract.
2. Using abstract class, we can group instances of related type together
3. Abstract class can extend only one abstract/concrete class.
4. We can define constructor inside abstract class.
5. Abstract class may or may not contain abstract method.

- **Hint :** In case of inheritance if state is involved in super type then it should be abstract.

Interface




```
Printable[] arr = new Printable[ 3 ];
arr[ 0 ] = new Date( ); //Upcasting
arr[ 1 ] = new Complex( ); //Upcasting
arr[ 2 ] = new Point( ); //Upcasting
```

Interface

1. If "is-a" relationship is not exist between super type and sub type and if we want same method design in all the sub types then super type must be interface.

2. Using interface, we can group instances of unrelated type together.

3. Interface can extend more than one interfaces.

4. We can not define constructor inside interface.

5. By default methods of interface are abstract.

- **Hint :** In case of inheritance if state is not involved in super type then it should be interface.

Thank you.

Rohan.paramane@sunbeaminfo.com