Agenda

- Association
- Inheritance
- super keyword
- Types of inheritance
- Method Overriding
- Upcasting & Downcasting
- Object class
- ~~Final~~
- ~~toString();~~
- ~~equals();~~

Association

- If "has-a" relationship exist between the types, then use association.
- To implement association, we should declare instance/collection of inner class as a field inside another class.
- There are two types of associations
    - 1. Composition
    - 2. Aggregation

Composition

- Represents part-of relation i.e. tight coupling between the objects.
- The inner object is essential part of outer object.
- Heart is part of Human.
- Engine is part of Car.
- Wall is part of Room.
- joining date is a part of employee

Aggegration

- Represents has-a relation i.e. loose coupling between the objects.
- The inner object can be added, removed, or replaced easily in outer object.
- Car has a Driver.
- Company has Employees.
- Room has a window
- Employee has a vehicle

Inheritance

- If "is-a"/"kind-of" relationship exist between the types, then use inheritance.
- Inheritance is process of generalization to specialization.
- All members of parent class are inherited to the child class.
- Parent class is also called as super class and child class is also called as sub-class.
- Example:

    - Manager is a Employee
    - Mango is a Fruit
    - Rectangle is a Shape
- In Java, inheritance is done using extends keyword.
- Java doesn't support multiple implementation inheritance i.e. a class cannot be inherited from multiple super-classes.
- However Java does support multiple interface inheritance i.e. a class can be inherited from multiple super interfaces.

Super Keyword

- In sub-class, super-class members are referred using "super" keyword.
- used for calling super class constructor
- By default, when sub-class object is created, first super-class constructor (param-less) is executed and then sub-class constructor is executed.
- "super" keyword is used to explicitly call super-class constructor.
- Super class members (non-private) are accessible in sub-class directly or using "this" reference. These members can also be accessed using "super" keyword.
- However, if sub-class method signature is same as super-class signature, it hides/shadows method of the super class i.e. super-class method is not directly visible in sub-class.
- The "super" keyword is mandetory for accessing such hidden members of the super-class.

Types of Inheritance

- 1. Single

```
class A {

}
class B extends A{

}
```

- 2. Multiple

```
class A {

}
class B {

}
class C extends A,B{ // Not Allowed

}

interface I1{

}
```

```
interface I2{

}

interface I3 extends I1,I2{ // Allowed

}

class D implements I1,I2{ // Allowed

}
```

- 3. Hirerachical

```
class A {

}
class B extends A{

}
class C extends A{

}
```

- 4. Multilevel

```
class A {

}
class B extends A{

}
class C extends B{

}
```

- Hybrid inheritance: Any combination of above types

Method Overriding

- Redefining a super-class method in sub-class with exactly same signature is called as "Method overriding".
- Programmer should override a method in sub-class in one of the following scenarios
    - 1. Super-class has not provided method implementation at all (abstract method).
    - 2. Super-class has provided partial method implementation and sub-class needs additional code. Here sub-class implementation may call super-class method (using super keyword).

    - 3. Sub-class needs different implementation than that of super-class method implementation.

Rules of method overriding in Java

- 1. Each method in Java can be overridden unless it is private, static or final.
- 2. Sub-class method must have same or wider access modifier than super-class method.
- 3. Arguments of sub-class method must be same as of super-class method.
- 4. The return-type of sub-class method can be same or sub-class of the super- class's method's return-type. This is called as "covarient" return-type.
- 5. Checked exception list in sub-class method should be same or subset of exception list in super-class method.

- If these rules are not followed, compiler raises error or compiler treats sub-class method as a new method.

- Java 5.0 added @Override annotation (on sub-class method) informs compiler that programmer is intending to override the method from the super-class.

- @Override checks if sub-class method is compatible with corresponding super-class method or not (as per rules). If not compatible, it raise compile time error.

- Note that, @Override is not compulsory to override the method. But it is good practice as it improves readability and reduces human errors.

Upcasting

- Assigning sub-class reference to a super-class reference.
- Sub-class "is a" Super-class, so no explicit casting is required.
- Using such super-class reference, only super-class methods inherited into sub-class can be called. This is "Object slicing".
- Using such super-class reference, super-class methods overridden into sub-class can also be called.

Downcasting

- Assigning super-class reference to sub-class reference.
- Every super-class is not necessarily a sub-class, so explicit casting is required.

```
Person p1 = new Employee();
Employee e1 = (Employee)p1; // down-casting - okay - Employee reference will point
to Employee object

Person p2 = new Person();
Employee e2 = (Employee)p2; // down-casting - ClassCastException - Employee
reference will point to Person object
```

Polymorphism

- poly = Many , morphism = Forms

- It has two types
    1. compile time
        - implemented using method overloading
        - Compiler can identify which method to be called at compile time depending on types of arguments. This is also referred as "Early binding".
    2. Runtime - implemented using method overriding - The method to be called is decided at runtime depending on type of object. This is also referred as "Late binding" or "Dyanmic method dispatch".

instanceof operator

- Java's instanceof operator checks if given reference points to the object of given type (or its sub-class) or not. Its result is boolean.
- Typically "instanceof" operator is used for type-checking before down-casting.

```
Person p = new SomeClass();
if(p instanceof Employee) {
    Employee e = (Employee)p;
    System.out.println("Salary: " + e.getSalary());
}
```

Object class

- Non final and non-abstract class declared in java.lang package.
- In java, all the classes (not interfaces) are directly or indirectly extended from Object class.
- In other words, Object class is ultimate base class/super class hierarchy.
- Object class is not inherited from any class or implement any interface.
- It has a default constructor. Object o = new Object();
- Object class methods (read docs)
    - public Object();
    - public native int hashCode();
    - public boolean equals(Object);
    - protected native Object clone() throws CloneNotSupportedException;
    - public String toString();
    - protected void finalize() throws Throwable;
    - public final native Class<?> getClass();
    - public final native void notify();
    - public final native void notifyAll();
    - public final void wait() throws InterruptedException;
    - public final native void wait(long) throws InterruptedException;
    - public final void wait(long, int) throws InterruptedException;