Day 11

Typing

- It is also called as polymorphism.
- polymorphism = poly(many) + morphism(forms/behavior).
- An ability of object/instance to use same interface to perform different behavior is called polymorphism.
- Its main job is to reduce maintenance of the system.
- Types of polymorphism:
    1. Compile time polymorphism
        - It is also called as static polymorphism/early binding/weak typing/false polymorphism.
        - We can achive it using method overloading.
    2. Run time polymorphism
    - It is also called as dynamic polymorphism/late binding/strong typing/true polymorphism.
    - In java Run time polymorphism is called as dynamic method dispatch.
    - We can achive it using method overriding.

Method Overloading

- If implementation of a method is logically same/eqivalant then we should give same name to the function.
- If we want to give same name to the method then we should use following rules:
    1. If type of parameters are same then number of paramter pass to the function must be different.

```
void sum( int num1, int num2 );
void sum( int num1, int num2, int num3);
```

```
2. If number of parameters are same then type of at least one parameter
must be different.
```

```
void sum( int num1, int num2 );
void sum( int num1, double num2 )
```

```
3. If number of parameters are same then order of type of parameters must
be different.
```

```
void sum( int num1, float num2 );
void sum( float num1, int num2 )
```

```
4. On the basis of different return type we can not overload method.
```

```
int sum( int num1, int num2 );
void sum( int num1, int num2 );
```

- Process of writing method using above rules is called method overloading and methods are called overloaded method.

Inheritance

- According to requirement, if implementation of exisiting method is logically incomplete then we should override/redine method in sub class.
- According to requirement, if implementation of exisiting class is logically incomplete then we should extend the class i.e we should use inheritance.
- Using super keyword, we can access members of super class inside method of sub class.

Types of inheritance

- Interface Inheritance
    1. Single Inheritance (Allowed in java)

```
interface A{    };
interface B extends A{    };
```

```
2. Multiple Inheritance (Allowed in java )
```

```
interface A{    };
interface B{    };
interface C extends A, B{    };
```

```
3. Hierarchical Inheritance(Allowed in java )
```

```
interface A{    }
interface B extends A{    }
interface C extends A{    }
interface D extends A {    }
```

```
4. Multilevel Inheritance(Allowed in java )
```

```
interface A{    }
interface B extends A{    }
interface C extends B{    }
interface D extends C {    }
```

- Implementation Inheritance
    1. Single Inheritance (Allowed in java)

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

```
2. Multiple Inheritance (Not Allowed in java )
```

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

```
3. Hierarchical Inheritance(Allowed in java )
```

```
class A{    }
class B extends A{    }
class C extends A{    }
class D extends A {    }
```

```
4. Multilevel Inheritance(Allowed in java )
```

```
class A{    }
class B extends A{    }
class C extends B{    }
class D extends C {    }
```

- In java, class can extend only one class.

```java
class Person    //class Person extends Object
{   }
```

- Here Object is direct super class of class Person

```java
class Person extends Object
{   }
class Employee extends Person
{   }
```

- Here Object is indirect super class of class Employee.
- Process of redining method of super class, inside sub class is called method overriding.
- During inheritance, members of super class inherit into sub class hence, sub class instance can be considered as super class instance.
- Since sub class instance can be considered as super class instance, we can use it in place of super class instance.

```java
Person p1 = new Person();   //OK
Person p2 = new Employee(); //OK
```

```java
Person p1 = new Person();   //OK
Person p2 = p1; //OK
```

```java
Employee emp1 = new Employee(); //OK
Person p2 = emp1;   //OK
```

- During inheritance, members of sub class do not inherit into super class hence, super class instance can not be considered as sub class instance.
- Since super class instance can not be considered as sub class instance, we can not use it in place of sub class instance.

```java
Employee emp1 = new Employee();  //OK
Employee emp2 = new Person();// Not OK
```

```java
Employee emp1 = new Employee(); //OK
Employee emp2 = emp1;   //OK
```

```java
Person p1 = new Person(); //OK
Employee emp1 = p1;    //Not OK
```

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

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

```java
Person p = new Employee(); //Upcasting
```

- using upcasting, we can reduce object depandacny in the code.
- Process of converting reference of super class into reference of sub class is called downcasting. Here explict typecasting is mandatory.

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

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

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

- Only in case of upcasting, we can do downcasting. Otherwise JVM will throw ClassCastException.