ណែនាំអោយស្គាល់ Polymorphism

Object Oriented Programming in Java

I. ដូចម្តេចទៅដែលហៅថា Polymorphism ?

ពាក្យថា Poly + morphism គឺសំដៅលើការប្រើប្រាស់ទំរង់ច្រើនទៅលើ Object Super Class និង លក្ខណៈរបស់ Methods បានច្រើនទំរង់ទាំងក្នុង Super Class និង Sub Class នៅក្នុងចំនុចនេះអ្នកនិងសិក្សាលើ៖

1. Method Overloading
2. Method Overriding
3. Types of Polymorphism – Runtime and compile time

1. Method Overloading: សំដៅលើការបង្កើតនូវ Methods ដែលមានឈ្មោះ ដូចគ្នា ចាប់ពីរឡើងទៅតែមានភាពខុសគ្នាលើ Return type function និង ចំនួន parameter របស់ function ទាំងនេះ។

Start Page | Employee.java

```
public class Employee {
    private String name;
    private String address;
    private int number;
    public Employee(String n,String a,int num)
    {
        name=n;
        address=a;
        number=num;
    }
    public void Print()
    {
     System.out.println (name + "    "  + address +  "    "  + number);
    }
     public String Print(String tel)
    {
         return (name + "    "  + address +  "    "  + number + "    "  + tel);
    }
    public int Print(int n1,int n2)
    {
     return this.number+n1+n2;
    }
     public static void main (String[] args) {
      Employee emp=new Employee("Sok Dara","Takeo",150);
      emp.Print();
      System.out.println ("Print=" + emp.Print("ETEC"));
      System.out.println ("Print=" + emp.Print(15,50));

        }

}
```

- Constructor Overloading: *សំដៅលើការបង្កើតនូវ* Constructor *ដែលមានឈ្មោះ ដូចគ្នា ចាប់ពីរឡើងទៅតែមានភាពខុសគ្នាលើ ចំនួន* parameter *របស់* Constructor *ទាំងនេះ។*

ឧទាហរណ៍ ៖

Start Page | Employee.java *

```
public class Employee {
    private String name;
    private String address;
    private int number;
    public Employee(String n,String a,int num)
    {
       name=n;
       address=a;
       number=num;
    }
      public Employee()
    {
       name="N/A";
       address="N/A";
       number=0;
    }
    public Employee(int num)
    {
       name="N/A";
       address="N/A";
       number=num;
    }
     public Employee(String n,int num)
    {
       name=n;
       address="N/A";
       number=num;
    }
    public void Print()
    {
     System.out.println (name + "     "  + address +  "     "  + number);
    }
     public static void main (String[] args) {
      Employee emp;
      emp=new Employee("Sok Dara","Takeo",150);
      emp.Print();
      emp=new Employee();
      emp.Print();
      emp=new Employee("Chan Love",50);
      emp.Print();
        }
}
```

2. Method Overriding

សំដៅលើការបង្កើតនូវ Method *ដែលមានឈ្មោះដូចគ្នាទាំងនោះនៅក្នុង* Super Class *និង* Sub Class *ពោលគឺ* Sub Class *អាចយក* Method *ដែល មានក្នុង* Super Class *មកបន្ថែមលក្ខណៈអោយវាថែម។*

ឧទាហរណ៍

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal {
   public void move() {
      super.move();   // invokes the super class method
      System.out.println("Dogs can walk and run");
   }
}
class Fish extends Animal {
   public void move() {
      super.move();   // invokes the super class method
      System.out.println("Fish can travel in water");
   }
}
public class TestDog {

   public static void main(String args[]) {
      Animal b = new Dog();   // Animal reference but Dog object
      b.move();   // runs the method in Dog class
      Animal b1 = new Animal();
      b1.move();
   }
}
```

- **Method Overriding:**

- ចំនួន Parameter ត្រូវតែដូចគ្នារវាង Class Super និង Sub Class
- Return type ត្រូវតែដូចគ្នារវាង Class Super និង Sub Class
- Method ជាលក្ខណៈ final និង Static មិនអាច overriding បានទេ
- Constructor មិនអាច Overriding បានទេៗ

3. Types of Polymorphism – Runtime and compile time

នៅក្នុង Concept របស់ polymophsim ត្រូវបានគេបែងចែក Concept ជាពីរប្រភេទទៀតគឺ៖

- Compile Time or Static polymorphism: សំដៅលើការធ្វើការនៅលើដំណាក់កាល Compile Code ជាលើកដំបូង។

```
Start Page | Employee.java ×

public class Employee {
    private String name;
    private String address;
    private int number;
    public Employee(String n,String a,int num)
    {
       name=n;
       address=a;
       number=num;
    }
    public void Print()
    {
     System.out.println (name + "    "  + address +  "    "  + number);
    }
     public String Print(String tel)
    {
        return (name + "    "  + address +  "    "  + number + "    "  + tel);
    }
    public int Print(int n1,int n2)
    {
     return this.number+n1+n2;
    }
     public static void main (String[] args) {
      Employee emp=new Employee("Sok Dara","Takeo",150);
      emp.Print();
      System.out.println ("Print=" + emp.Print("ETEC"));
      System.out.println ("Print=" + emp.Print(15,50));

        }

}
```

Overloading method គឺជាប្រភេទ method ដែលពិពណ៌នាលើការប្រើប្រាស់លក្ខណៈជា Static Polymophsim ដោយវាគ្រាន់តែជ្រើសរើសនូវ Function ណាមួយមានលក្ខណៈនៃចំនួន Parameter និង ប្រភេទទិន្នន័យខុសគ្នាមកធ្វើការជាការស្រេចនៅក្នុងពេលដែលអ្នកបាន Compile នៃCode នោះ។

- Runtime Polymophsim: *គឺជាប្រភេទ* Concept *មួយបែបទៀតដែលវាធ្វើការក្នុងដំណាក់កាល* Run Code ។ *ក្នុងចំនុចនេះគឺត្រូវជ្រើសរើសនូវ* Cocept Overring Method *មកពន្យល់ពីលក្ខណៈទាំងអស់នេះ*

```
class Test1{
    protected int x;
    protected int y;
    public Test1(int x,int y){
        this.x=x;
        this.y=y;
    }
    public void Print()
    {System.out.println ("X=" + x + "    Y=" + y + "\n");
    }
}
class Test2 extends Test1{
    protected int z;
  public Test2(int x,int y,int z)
    {
       super(x,y);
       this.z=z;
    }
    public void Print()
    {
        System.out.println ("X=" + x + "    Y=" + y + "  Z=" + z);
    }
}
class Test3 extends Test1{
    protected int z;
    public Test3(int x,int y,int z)
    {
       super(x,y);
       z=z;
    }
    public void Print()
    {
        System.out.println ("X=" + x + "    Y=" + y + "  Z=" + z);
    }
}
class Employee{
public static void main (String[] args) {
  Test1 obj;
  //Calling Method in Super Class
  obj=new Test1(100,200);
  obj.Print();
  //Calling Method in Sub Class1
  obj=new Test2(100,200,300);
  obj.Print();
  obj=new Test3(10,20,30);
  obj.Print();
}
```

II. ដូចម្តេចទៅដែលហៅថា Abstract Class ?

Abstract Class គឺជាប្រភេទ Class ដែលបង្កើតឡើងដោយ Keyword Abstract ហើយត្រូវតែ មានន្ទូវ method Abstract មួយយ៉ាងតិច។

Method Abstract គឺជាប្រភេទ Method ដែលមានតែឈ្មោះ ប្រកាសតែគ្មានខ្លួន។

គេបង្កើតន្ទូវ Class Abstract និង Method Abstract សំរាប់អោយ Sub Class អាច Extend ទៅ ប្រើ និង override លើ Method ដែលមានស្រាប់ក្នុង Abstract Classនោះ។

ឧទាហរណ៏ ១៖

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

លទ្ធផលទទួលបាន៖

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

ឧទាហរណ៍ ២៖

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

លទ្ធផលទទួលបាន៖

```
bike is created
running safely..
gear changed
```

ចូរបង្កើតនូវ Class ដូចខាងក្រោម៖

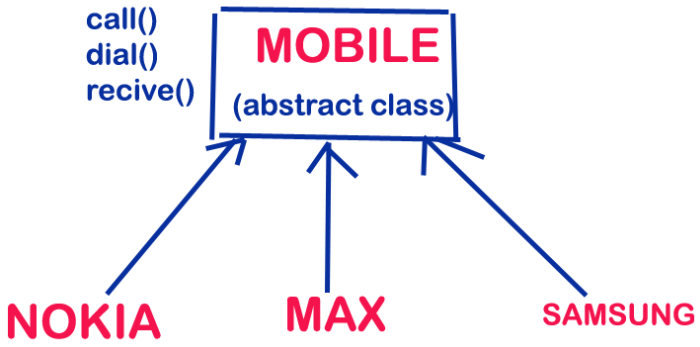

fig: implementing common features to sub classes

ឧទាហរណ៍ ១៖ ការបង្កើតនូវ Interface សំរាប់អោយ Abstract class Implement មកកាន់វា។

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){System.out.println("I am c");}
}

class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

```
Output:I am a
       I am b
       I am c
       I am d
```

គេមាននូវ Interface មួយដូចខាងក្រោម ចូរបង្កើតនូវ Abstract class មួយឈ្មោះ DemoSport ដែល Implement ទៅ កាន់ interface Hockey និង Football ហើយបង្កើតនូវ Object ពី Abstract Class យក មកប្រើប្រាស់។

```
// Filename: Sports.java
public interface Sports {
   public void setHomeTeam(String name);
   public void setVisitingTeam(String name);
}

// Filename: Football.java
public interface Football extends Sports {
   public void homeTeamScored(int points);
   public void visitingTeamScored(int points);
   public void endOfQuarter(int quarter);
}

// Filename: Hockey.java
public interface Hockey extends Sports {
   public void homeGoalScored();
   public void visitingGoalScored();
   public void endOfPeriod(int period);
   public void overtimePeriod(int ot);
}
```