How are you?

27th Feb 2024

Date: __27th February 2024__Face2Face___

Recap

- Analysis !!!
  - 1. Better to produce an SRS ➔ simplified to a flow chart, use case diagram or any other Behavioral Diagrams
  - 2. Identify the objects, classes (properties and methods),
- Implement/code the classes and have them interact by invoking methods across as per the analysis
- Classes implementation
  - Main method
  - Other methods (functionalities)
  - Constructors!
- Inheritance

```
package tuk;
/**
 * @author Salesio
 */
public class TUKGateSystem{
    /**
     This class is the Application entry
       point*/
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("I am the main method");
        Person person1 = new Person("Salesio", 2);
        System.out.println(person1.password);
    }
}
```

```
package tuk;
// @author Salesio
public class Person {
    int id;
    String name;
    String password;
    public Person(int anId, String aName){
        id=anId;
        name = aName;
      }
}
```

What is the Output?

What is the Output, now?

```
package tuk;
/**
 * @author Salesio
 */
public class TUKGateSystem{
   /**
    This class is the Application entry
      point*/
   public static void main(String[] args) {
      // TODO code application logic here
      System.out.println("I am the main method");
      Person person1 = new Person(2,"Salesio");
      System.out.println(person1.password);
   }
}
```

```
package tuk;
// @author Salesio
public class Person {
   int id;
   String name;
   String password;
   public Person(int anId, String aName){
      id=anId;
      name = aName;
password = Security().createPassword(id, name);
     } }
```

```
package tuk;
/** @author Salesio */
class Security {
    String createPassword(int id, String name) {
        String pass = id+name.substring(1);
        return pass;
    }
}
```

- Why would having two or more classes above make sense?
  - Easy maintenance!
  - E.g. assume that the rules of generating usernames change. We only have to change and compile the class that generates Ids/Usernames

Overriding

- What does it mean **Overriding** in English
- ? “is over …” ?

Overloading

- What does it mean **Overloading** in English
- ? “is more …” ?

Overloading Methods

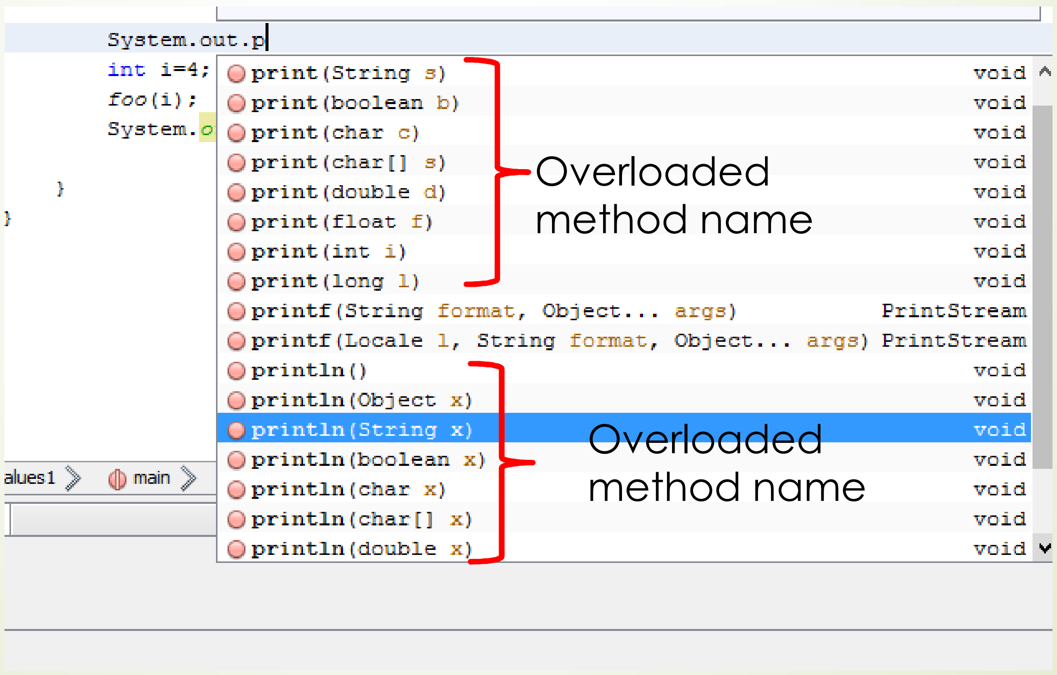
- Two methods have the same name but different parameter lists within one class. The Java compiler determines which method is used based on the method signature.

```
public static double max(double num1, double num2) {
        if (num1 > num2)
        return num1;
        else
        return num2;
 }
```

Consider these methods in the same class

```
class mathematics {
    public static int max(int num1, int num2) {
        int result;
        if (num1 > num2)
          result = num1;
        else
          result = num2;
        return result;
      }
    public static double max(double num1, double num2) {
        if (num1 > num2)
              return num1;
            else
              return num2;
          }
    public static double max(double num1, double num2) {
                if (num1 > num2)
                return num1;
                else
                return num2;
          }
    }
```

- If you call max with int parameters, the max method that expects int parameters will be invoked; if you call max with double parameters, the max method that expects double parameters will be invoked.
- Example of overloading in the java API



```
System.out.p
int i=4;
foo(i);
System.o
    }
}
```

```
print(String s)                                    void
print(boolean b)                                   void
print(char c)                                      void
print(char[] s)                                    void
print(double d)                                    void
print(float f)                                     void
print(int i)                                       void
print(long l)                                      void
printf(String format, Object... args)       PrintStream
printf(Locale l, String format, Object... args) PrintStream
println()                                          void
println(Object x)                                  void
println(String x)                                  void
println(boolean x)                                 void
println(char x)                                    void
println(char[] x)                                  void
println(double x)                                  void
```

Method Overloading

- Method overloading in Java refers to the ability to define multiple methods within a class with the same name but with different parameter lists. i.e. methods with the same name can have different numbers or types of parameters, allowing the programmer to use the same method name for different behaviors or functionalities based on the parameters passed.
- Note the following key points with overloading:
  - Same Method Name: Overloaded methods have the same name but different parameter lists.
  - Parameter Lists: The parameter lists of overloaded methods must differ in at least one of the following ways:
    - Number of parameters
    - Data types of parameters
    - Order of parameters

Overriding methods

- Definition:
  - In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
  - When a method in a subclass has the same ***name***, same ***parameters*** or **signature**, and same **return type**(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
- Uses
  - to provide the specific implementation of a method which is already provided by its superclass
  - To implement runtime polymorphism - If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed
- REF:
  - https://pragjyotishcollege.ac.in/wp-content/uploads/2020/04/Method-Overriding-in-Java.pdf

- Implementation examples
  - Ideas in groups
    - 3
  - Implementations

- Thanks

Polymorphism

Polymorphism

- "poly" means many and "morphs" means forms. So polymorphism means many forms.
- Polymorphism is not a programming concept but it is one of the principal concepts of OOP.
- Polymorphism is common in OOP languages but is implemented differently

Dynamic or runtime polymorphism

- Is achieved via method overriding (in inheritance)
- Reference of *parentclass* (e.g. Person) can hold object of the *parentclass* (i.e Person) or an object of any of its subclass (e.g. Staff)
- E.g. the following statements are valid:
  - Person obj = new Staff(); // parent class reference can be assigned to child object
- Since in method overriding both the classes(parent class and child class) have same method, compiler doesn't figure out which method to call at compile-time. In this case JVM(java virtual machine) decides which method to call at runtime that's why it is known as runtime or dynamic polymorphism.

Example

```
public class Person {
   int id;
   String name;
   public Person(int anId, String aName){
      id=anId;
      name = aName;
     }
  public void printDetails(){
     System.out.println("ID: "+id);
     System.out.println("Name: "+name);
    }
}
```

```
public class Staff extends Person{
   String staffId;
   Staff(String aStaffId, String staffName,
               int nationalId)
    {
      super(nationalId, staffName);
      this.staffId = aStaffId;
    }
   public void printDetails(){
      super.printDetails();
      System.out.println("StaffPayroll: "
          +staffId);
      }
}
```

```
public class HIS {
public static void main(String[] args) {
Person pers1 = new Person(12345,"Haile");
      Person pers2 = new Staff("FA2125", "Jonathan", 123234);
      pers2.printDetails();
      System.out.println("---------------------");
      pers1.printDetails();
   }
}
```

Static or Compile time Polymorphism

- This is achieved via method overloading
- A class has more than one methods with the same methods '***sharing***' the same name but different number of arguments or different types of arguments or both
- When the method is called on an object, it is the parameters used in the call that determines what method code is executed
- Method overloading is an example of compile-time polymorphism (also known as static polymorphism) because the compiler determines which method to call based on the method signature at compile time.

```
class X
{
  void methodA(int num) {
     System.out.println ("methodA:" + num);
  }
  void methodA(int num1, int num2){
     System.out.println ("methodA:" + num1 + ","
+ num2);
  }
  double methodA(double num) {
     System.out.println("methodA:" + num);
     return num;
  }
}
```

```
class Y{
  public static void main (String args []){
     X Obj = new X();
     double result;
     Obj.methodA(20);
     Obj.methodA(20, 30);
     result = Obj.methodA(5.5);
     System.out.println("Answer is:" + result);
  }
}
```

Output?:

```
class X
{
  void methodA(int num) {
     System.out.println ("methodA:" + num);
  }
  void methodA(int num1, int num2){
     System.out.println ("methodA:" + num1 + ","
+ num2);
  }
  double methodA(double num) {
     System.out.println("methodA:" + num);
     return num;
  }
}
```

```
class Y{
  public static void main (String args []){
     X Obj = new X();
     double result;
     Obj.methodA(20);
     Obj.methodA(20, 30);
     result = Obj.methodA(5.5);
     System.out.println("Answer is:" + result);
  }
}
```

Output (answer):

```
methodA:
methodA:
methodA:
Answer is:
```

```
class X
{
  void methodA(int num) {
     System.out.println ("methodA:" + num);
  }
  void methodA(int num1, int num2){
     System.out.println ("methodA:" + num1 + ","
+ num2);
  }
  double methodA(double num) {
     System.out.println("methodA:" + num);
     return num;
  }
}
```

```
class Y{
  public static void main (String args []){
     X Obj = new X();
     double result;
     Obj.methodA(20);
     Obj.methodA(20, 30);
     result = Obj.methodA(5.5);
     System.out.println("Answer is:" + result);
  }
}
```

Output (answer):

```
methodA:20
methodA:20,30
methodA:5.5
Answer is:5.5
```

Do we have Polymorphism here? Explain.

Example Question

Given the case:

- In the context of the ongoing digitalization of TU-K, the University head of Security has approached you to design a simple system for the askaris manning the University gate. The idea is to automate / computerize what happens at the gate. Starting with the Visitors-Book. The visitors book records all visitors to the University premises. At any given gate, a record of a visitor shows, among other things the following:
- The officer who attended to the visitor, the date of visit, details of the visitor, destination point/office, the purpose/objective of the visit, mode of traveling used (and corresponding details), report from the visited person, gate used to exit, etc.
- After listening to the visitor, the security officer (askari) must indicate on the form whether the purpose of visit is official, private, or returning resident. Official can be administrative office visit, lecturing, studying, working, etc

Required:

A. Explain the overall importance of Polymorphism in good Software Engineering practice

B. Using java you are required to demonstrate the concept of Polymorphism. Your code must include comments that clearly explains how the various constructors, methods and/or statements implement Polymorphism

Next

- personal work implementations of what we have covered above (Be unique and not just using examples from the books and notes)
- Methods, Java parameter passing
- mini project (initial Assessment)

- Thank you