Inheritance

Providing a unique form of code-sharing

Alwin Tareen

What is Inheritance?

Code sharing

- Inheritance provides a unique form of code-sharing by allowing you to take the implementation of any class, and build a new class based on that implementation.

Subclasses and superclasses

- A **subclass** starts by inheriting all of the public data and methods that are defined in the **superclass**.
- The subclass can then extend its behavior by adding additional data and new methods.
- The subclass can also extend or replace behavior in the superclass by **overriding** methods that were already implemented.

What is Inheritance?

The **is-a** relationship

- Inheritance implements the **is-a** relationship between objects:

```
subclass  is-a  superclass
high school  is-a  school
student  is-a  person
```

Rules for Subclasses

A subclass can add new `private` instance variables.

- The `Student` class adds two `private` instance variables of its own: `ID` and `classification`.

A subclass can add new `public` or `private` methods.

- The `Student` class inherits the methods `getName`, `getAddress`, `getPhone`, and `toString` from the `Person` class.
- The `Student` class adds two methods of its own: `getID` and `getClassification`.

Rules for Subclasses

A subclass can override(redefine) inherited methods.

- The `Person` class defines a `toString` method. The `Student` class also defines a `toString` method with the exact same signature(same name, return type and quantity/data type of parameters).
- Therefore, the `Student` class has overridden or redefined the `toString` method, so that its behavior is more suitable to its specific needs.

Rules for Subclasses

Partial overriding

- Sometimes the code for overriding a method includes a call to the superclass method. This is called partial overriding.
- Typically, this occurs when the subclass method wants to do what the superclass method does, plus something extra.
- This is achieved by using the keyword `super` in the implementation.
- The `toString` method in the `Student` class partially overrides the `toString` method in the `Person` class.
- The statement `super.toString()` signals that the `toString` method in the superclass should be invoked here.

Rules for Subclasses

A subclass must define its own constructors.

- Constructors are **not** inherited, therefore a subclass has to provide its own.

A subclass cannot directly access the `private` members of its superclass.

- A subclass should use the publicly declared accessor/mutator methods to access the instance variables of its superclass.

Rules for Subclasses

A subclass' constructors can explicitly call the superclass' constructors by using the keyword `super`.

- If `super` is used, then it must be the `first` statement in the subclass' constructor.
- For example, the `Student`'s constructor calls the `Person` class' constructor with the statement: `super(n, a, p);`
- This allows the `name`, `address` and `phone` variables to be initialized.

The Person Class

```
public class Person
{
    private String name;
    private String address;
    private String phone;

    public Person(String n, String a, String p)
    {
        name = n;
        address = a;
        phone = p;
    }

    public String getName()
    {
        return name;
    }
```

The Person Class, Continued

```
    public String getAddress()
    {
        return address;
    }

    public String getPhone()
    {
        return phone;
    }

    public String toString()
    {
        String result = "Name: " + name + "\n";
        result += "Address: " + address + "\n";
        result += "Phone: " + phone + "\n";
        return result;
    }
}
```

The Student Class

```
public class Student extends Person
{
    private String ID;
    private int classification;

    public Student(String n, String a, String p, String id, int c)
    {
        super(n, a, p);
        ID = id;
        classification = c;
    }

    public String getID()
    {
        return ID;
    }

    public int getClassification()
    {
        return classification;
    }
```

The Student Class, Continued

```
    public String toString()
    {
        result = super.toString() + "\n";
        result += "ID: " + ID + "\n";
        result += "classification: " + classification;
        return result;
    }
}
```

The PersonTest Class

```
public class PersonTest
{
    public static void main(String[] args)
    {
        Person bob = new Person("Bob", "BDNS", "3245893");
        System.out.println(bob);
        System.out.println();

        Student dan = new Student("Dan", "BNDS", "8675309", "SN938", 4);
        System.out.println(dan);
    }
}
```

Inheritance: End of Notes