Module 1 Day 11

Inheritance

Module 1 Day 11

Can you?

1. Describe the purpose and use of inheritance

2. Given an existing set of classes, properly identify the subclasses of superclasses

3. Define and use superclasses and subclasses in an inheritance hierarchy

Inheritance: The “Is A” relationship

Real world objects exhibit parent-child relationships. Consider the following examples:

- Humans, dogs, elephants, and whales are clearly quite different from each other, but they have enough similarities prompting biologists to classify them as mammals.

- Cars, motorcycles, and trucks are all motor vehicles, but they each have sufficient differences that the Department of Motor Vehicles regulates and licenses them in different ways.

- In finance, related funds are held in accounts, yet an account can refer to a checking account, savings account, or mutual fund. As accounts, they all share similarities like a monthly balance and account holder name.

Inheritance

Java and other modern programming languages provide powerful tools that enable developers to model parent-child relationships.

- In Java one class can be classified as a child of a parent class.
  a. The child class will then inherit the properties and methods defined in the parent.

- Inheritance can defined in several ways; by:
  a. a **concrete class** (all the classes we have seen so far) inheriting from another concrete class by ***extending*** that class.
  b. A concrete class inheriting from an **Interface** by ***implementing*** that interface. (this will be covered on Day 12)
  c. a concrete class inheriting from an **abstract class** by ***extending*** that abstract. (this will be covered on Day 13)

Inheritance: Declaration

In this module, we will explore the situation where two concrete classes have a parent child relationship. A child class that will inherit from a parent must be defined following this syntax:

public class **<<Name of Child Class>>** **extends** **<<Name of Parent Class>>** {

… // rest of your class declaration

}

Inheritance Example: A Car Is-A Vehicle

Vehicle has defined several methods and data members. In this example, Vehicle serves as the parent class.

```
package te.mobility;

public class Vehicle {

        private int numberOfWheels;
        private double engineSize;
        private String bodyColor;

        public int getNumberOfWheels() {
                return numberOfWheels;
        }
        public void setNumberOfWheels(int numberOfWheels) {
                this.numberOfWheels = numberOfWheels;
        }
}
```

Car is a child class of Vehicle.. Note how it is able to call Vehicle's methods. The extends syntax is used to create this relationship.

```
package te.mobility;

public class Car extends Vehicle {

        public void report() {
                System.out.println(super.getNumberOfWheels());
                // 0, inherited from parent class which will have the
                // default value for integers.

                super.setNumberOfWheels(4);
                // we are calling the setter defined on its parent

                System.out.println(super.getNumberOfWheels());
                // 4
        }
}
```

Inheritance Example: A Truck Is-A Vehicle Too

Here we define another child class of Vehicle called Truck.

```
package te.mobility;

public class Truck extends Vehicle {

        public void report() {
                super.setNumberOfWheels(10);
                // we are calling the setter defined on its parent
        }

        public void coupleCargoContainer() {
                System.out.println("coupling cargo container");
                super.setNumberOfWheels(18);

        }
}
```

We now have a parent class with 2 child classes. Vehicle is the parent class. Both Truck and Car extends from vehicle, making them child classes of the vehicle.

Inheritance Example

```
package te.main;

import te.mobility.Car;
import te.mobility.Truck;

public class Garage {

        public static void main(String args[]) {

                Car myCar = new Car();
                myCar.setNumberOfWheels(4);
                System.out.println(myCar.getNumberOfWheels());

                Truck myTruck = new Truck();
                myTruck.setNumberOfWheels(10);
                System.out.println(myTruck.getNumberOfWheels());
                myTruck.coupleCargoContainer();
                System.out.println(myTruck.getNumberOfWheels());

                // This is an invalid call:
                //myCar.coupleCargoContainer();
}}
```

Suppose have an application class called Garage with a main method that will instantiating new cars and trucks based on the inheritance we've defined so far.

Output will be 4

Output will be 10

Output will be 18

This is an invalid statement, the coupleCargoContainer method is unique to the Truck class.

Effect of Private Modifiers on Inheritance

The access modifiers present on the parent class' data members is not trivial.

- Data members and methods marked as private on a parent class cannot be inherited by a child class.

- Data members and methods marked as protected can be inherited by a child class even if it's on a different package.

Effect of Private Modifiers on Inheritance

Consider the following example:

```
package te.mobility;

public class Vehicle {
...
        private String privateMethod() {
                return "private";
        }
…
}
```

We are assuming that the Car class extends from Vehicle like on the previous examples.

```
package te.main;

import te.mobility.Car;
import te.mobility.Truck;

public class Garage {

        public static void main(String args[]) {

                Car myCar = new Car();
                myCar.setup();
                myCar.privateMethod();
                …
        }
}
```

This is an invalid call.

Constructors & Inheritance: Invoking the Parent

If a Parent has implemented a constructor, a Child class must add a call using super(...) to invoke the parent's constructor with the correct arguments.

The syntax of super(...) is as follows:

```
public ChildClass(<<argument 1>>, <<argument2>>, ….) {

    super(<<argument1>>, <<argument2>>, ...);

}
```

Here the Child arguments listed are arguments for the Parent's constructor

Constructors & Inheritance: Example

```
package te.mobility;

public class Vehicle {

        private int numberOfWheels;
        private double engineSize;
        private String bodyColor;

        public Vehicle(int numberOfWheels, double engineSize, String bodyColor) {
                this.numberOfWheels = numberOfWheels;
                this.engineSize = engineSize;
                this.bodyColor = bodyColor;
        }

        …
}
```

There is now a constructor in the parent Vehicle class.

Invoking the Parent Constructor: Example

Truck will now have to implement a constructor with a super(...) call.

```
public class Truck extends Vehicle {

        public Truck(int numberOfWheels, double engineSize, String bodyColor) {
                super(numberOfWheels, engineSize, bodyColor);
        }
...
}
```

```
public class Vehicle {
...
        public Vehicle(int numberOfWheels, double engineSize, String bodyColor) {
                this.numberOfWheels = numberOfWheels;
                this.engineSize = engineSize;
                this.bodyColor = bodyColor;
        }
        …
}
```

The super(...) call is a call to the parent constructor with the matching signature.

Constructors on Parent Classes: Example

In the Garage orchestrator class we can now instantiate a new Truck with the constructor.

```
package te.main;

import te.mobility.Truck;

public class Garage {

	public static void main(String args[]) {

		Truck cargoTruck = new Truck(10, 14.8, "red");
	}
}
```

Multiple Constructors

Classes can contain more than one constructor, each taking a different number of arguments.

Multiple Constructors Example

```
public class Vehicle {

        private int numberOfWheels;
        private double engineSize;
        private String bodyColor;

        public Vehicle(int numberOfWheels, double engineSize, String bodyColor) {
                this.numberOfWheels = numberOfWheels;
                this.engineSize = engineSize;
                this.bodyColor = bodyColor;
        }

        public Vehicle(int numberOfWheels, double engineSize) {
                this.numberOfWheels = numberOfWheels;
                this.engineSize = engineSize;
        }
…
}
```

Multiple Constructors Example

```
public class Truck extends Vehicle {

        public Truck(int numberOfWheels, double engineSize, String bodyColor) {
                super(numberOfWheels, engineSize, bodyColor);
        }

        public Truck(int numberOfWheels, double engineSize) {
                super (numberOfWheels, engineSize);
        }
```

The child class has also implemented a matching second constructor and called the 2 argument parent constructor using super.

Inheritance Facilitates Polymorphism!

If a Car Is-A Vehicle and a Truck Is-A Vehicle, can it be said that a Vehicle takes on many forms?

If a Car Is-A Vehicle and a Truck Is-A Vehicle, can both Cars and Trucks be treated as Vehicles in a generic or abstract sense?

Let’s find out…


... To The Code!