Module 1-11

Inheritance

Inheritance

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

- Humans, dogs, elephants, and whales are clearly quite different from each other, but they are all mammals.
- Cars, motorcycles, and trucks are all motor vehicles, but they each have sufficient differences for the DMV to regulate them differently.
- In finance, the word account can refer to a checking account, a savings account, or mutual fund, but they all share similarities like a monthly balance and account holder name.

Inheritance

We can also think of these relationships in terms of “is-a”:

- A human is a mammal, a dog is a mammal, a whale is a mammal…
- A truck is a motor vehicle, so is a car…

Inheritance: Declaration

Here we have the basic declaration to express this is-a relationship:

```
public class Name of Child Class extends Name of Parent Class {

… // rest of your class declaration

}
```

Inheritance Example

Vehicle has defined several methods and fields. 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;
        }
}
```

We use the **super** keyword to refer to the parent's members and variables.

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

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 coupeCargoContainer() {
                System.out.println(“...convoy!”);
                super.setNumberOfWheels(18);
        }
}
```

Let’s create another child class of Vehicle, this time Truck.

The Truck class has its own unique method, it has a method called coupleCargoContainer() which is unique to the Truck class, and not part of the Vehicle or Car class.

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();
                System.out.println(myCar.getNumberOfWheels());

                Truck myTruck = new Truck();

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

Suppose there is a class called Garage with a main method that will instantiating new cars and trucks..

The highlighted code will not compile since coupleCargoContainer() 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 on Parent Classes

If a parent has implemented a constructor, a child class must add a call using super(...). The syntax of super(...) is as follows:

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

    super(argument1, argument2, ...);

}
```

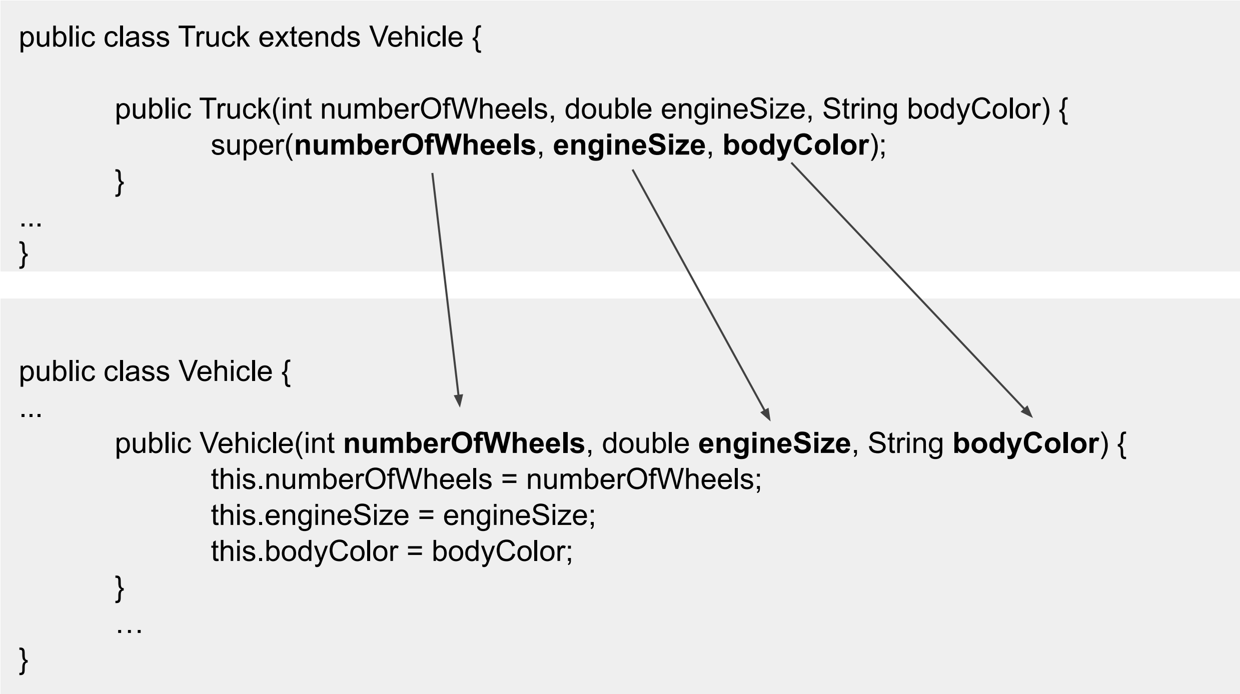
Constructors on Parent Classes: Example

We have declared a constructor for Vehicle:

```
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;
    }

}
```

Constructors on Parent Classes: Example

Note how the child class, 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 basically a call to the parent constructor, providing any required parameters

Constructors on Parent Classes: Example

In the Garage orchestrator class note how we are able to 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;
        }
…
}
```

Note that there is now a second constructor that does not take a bodyColor argument.

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);
        }
```

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

Method Overriding

- A subclass can provide a different implementation of a parent’s method.
- This is known as **method overriding.**

Method Overriding Example

```
public class ParentClass {

        public void sing() {
                System.out.println("I've been for a walk.");
        }
}
```

```
public class ChildClass extends ParentClass {

        @Override
        public void sing() {
                System.out.println("On a winter's day.");
        }
}
```

```
public class Song {

        public static void main(String[] args) {

                ParentClass parent = new ParentClass();
                ChildClass child = new ChildClass();

                parent.sing();
                //prints ParentClass's version: I've been for a walk.

                child.sing();
                //prints ChildClass's version: On a winter's day.

        }
}
```

FYI: The Object class

Java is built almost entirely on a series of is-a inheritance relationships, all classes can be traced back a class called **Object**.