CSE201: Monsoon 2020, Section-A
Advanced Programming

Lecture 07: Constructors in Inheritance Tree

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

This Lecture

- Constructor invocation in inheritance tree

Slide acknowledgements: CS15, Brown University

Indirectly Accessing `private` Instance Variables in Superclass by defining Accessors and Mutators

```
public class Car  {

    private Radio _myRadio;

    public Car() {
        _myRadio = new Radio();
    }

    protected Radio getRadio(){
        return _myRadio;
    }
    protected void setRadio(Radio radio){
        _myRadio = radio;
    }

}
```

- Remember from earlier that private variables are not directly inherited by subclasses
- If `Car` does want its subclasses to be able to access and change the value of `_myRadio`, it can define protected accessor and mutator methods
  - Will non-subclasses be able to access `getRadio()` and `setRadio()` ?
- Very carefully consider these design decisions in your own programs – which properties will need to be accessible to other classes?

Calling Accessors/Mutators From Subclass

- `Convertible` can get a reference to `_radio` by calling `this.getRadio()`
  - Subclasses automatically inherit these public accessor and mutator methods
- Note that using "double dot" we've chained two methods together
  - First, `getRadio` is called, and returns the radio
  - Next, `setFavorite` is called on that radio

```
public class Convertible extends Car {
   public Convertible() {
   }

   public void setRadioPresets(){
      this.getRadio().setFavorite(1, 95.5);
      this.getRadio().setFavorite(2, 92.3);
   }
}
```

Let's step through some code

- Somewhere in our code, a `Convertible` is instantiated

```
//somewhere in the program
Convertible convertible = new Convertible();
convertible.setRadioPresets();
```

- The next line of code calls `setRadioPresets()`
- Let's step into `setRadioPresets()`

Let's step through some code

- When someone calls `setRadioPresets();` first line is `this.getRadio()`
- `getRadio()` returns `_myRadio`
- What is the value of `_myRadio` at this point in the code?
  - Has it been initialized?
  - Nope, assuming that the structure of class Car is exactly as shown on right side (i.e. without any constructor), we'll run into a `NullPointerException` here :(

```
public class Convertible extends Car {
   public Convertible() { //code elided
   }

   public void setRadioPresets() {
      this.getRadio().setFavorite(1, 95.5);
      this.getRadio().setFavorite(2, 92.3);
   }
}
```

```
public class Car {

   private Radio _myRadio;

   public Radio getRadio() {
      return _myRadio;
   }
}
```

Making Sure Superclass's Instance Variables are Initialized

- `Convertible` may declare its own instance variables, which it initializes in its constructor
- `Car`'s instance variables are initialized in the `Car` constructor
- When we instantiate `Convertible`, how can we make sure `Car`'s instance variables are initialized too?
  - Case-1: Car has a default constructor that instantiate all its fields
  - Case-2: Car has a parameterized constructor for initializing all its fields

`super()`: Invoking Superclass's Default Constructor (Case 1)

- Let's assume that `Car`'s instance variables (like `_radio`) are initialized in `Car`'s default constructor
- Whenever we instantiates `Convertible`, default constructor of `Car` is called automatically
- To **explicitly** invoke `Car`'s default constructor, we can call `super()` inside the constructor of `Convertible`
  - **Can only make this call once**, and it must be the very first line in the subclass's constructor

```
public class Convertible extends Car {

   private ConvertibleTop _top;

   public Convertible() {
      super();
      _top = new ConvertibleTop();
      this.setRadioPresets();
   }

   public void setRadioPresets(){
      this.getRadio().setFavorite(1, 95.5);
      this.getRadio().setFavorite(2, 92.3);
   }
}
```

super(): Invoking Superclass's Parameterized Constructor (Case 2)

```
public class Car {

    private Racer _driver;
    public Car(Racer driver) {
        _driver = driver;
    }
    ......
}
```

```
public class Convertible extends Car {

    private ConvertibleTop _top;

    public Convertible(Racer driver) {
        super(driver);
        _top = new ConvertibleTop();
    }

    ......
}
```

- What if the superclass's constructor takes in a parameter?
  - We've modified `Car`'s constructor to take in a `Racer` as a parameter
  - How do we invoke this constructor correctly from the subclass?
- In this case, need the `Convertible`'s constructor to also take in a `Racer`
- The `Racer` is then passed as an argument to `super()` – now `Racer`'s constructor will initialize `_driver` to the instance of `Racer` that was passed to the `Convertible`

What if we don't call `super()`?

- What if we forget to call `super()`?
- If you don't explicitly call `super()` first thing in your constructor, Java automatically calls it for you, passing in no arguments
- But if superclass's constructor requires a parameter, you'll get an error!
- In this case, we get a **compiler error** saying that there is no constructor "`public Car()`", since it was declared with a parameter

```
public class Convertible extends Car {

    private ConvertibleTop _top;

    public Convertible(Racer driver) {
        //oops forgot to call super()
        _top = new ConvertibleTop();
    }

    ......
}
```

Next Lecture

- Abstract class
- Immutable class