More About Inheritance

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim   jwhh.com

Overview

Special reference: super

Preventing class inheritance

Preventing method overriding

Requiring class inheritance

Requiring method overriding

Constructors and inheritance

Special Reference: super

Similar to the special reference this

- Refers to the current object

Has a key difference from this

- Treats as an instance of the base class
- Provides access to overridden base class members

Flight.java

```java
public class Flight {
  private int flightNumber;
  private char flightClass;

  @Override
  public boolean equals(Object o) {
    if (!(o instanceof Flight))
        return false;

    Flight flight = (Flight) o;

    return flightNumber == flight.flightNumber &&
            flightClass == flight.flightClass;
  }
} // other members elided
```

Special Reference: super

```
Flight f1 = new Flight(175);

Flight f2 = f1;

// do some other stuff

if(f1.equals(f2))

  // do something
```

Flight.java

```java
@Override
  public boolean equals(Object o) {
    if(equals(o))
        return true;
    if (!(o instanceof Flight))
        return false;
    Flight flight = (Flight) o;
    return flightNumber == flight.flightNumber &&
            flightClass == flight.flightClass;
  }
```

Flight.java

```java
@Override
  public boolean equals(Object o) {
    if(equals(o))
        return true;

    if (!(o instanceof Flight))
        return false;

    Flight flight = (Flight) o;

    return flightNumber == flight.flightNumber &&
            flightClass == flight.flightClass;

  }
```

Flight.java

```java
@Override
  public boolean equals(Object o) {
    if(super.equals(o))
        return true;

    if (!(o instanceof Flight))
        return false;

    Flight flight = (Flight) o;

    return flightNumber == flight.flightNumber &&
            flightClass == flight.flightClass;

  }
```

Preventing Inheritance and Method Overriding

Default inheritance behavior

- Each class can be extended
- Derived class can override any method

Can change default behavior with final

- Can prevent class extending
- Can prevent method overriding

Preventing Inheritance

```
public class Passenger {
 // ...
}
```

Preventing Inheritance

```
final public class Passenger {
  // ...
}
```

Preventing Inheritance

```
public final class Passenger {
  // ...
}
```

Preventing Method Overriding

```
class CargoFlight extends Flight {

  public final void add1Package(float h, float w, float d) { ... }

  private boolean hasCargoSpace(float size) { ... }
  private void handleNoSpace() { ... }

} // other members elided
```

Requiring Inheritance and Method Overriding

Default class usage

- Each class can be directly instantiated

Default method overriding requirements

- Derived class has option whether to override a method

Can change default behavior with abstract

- Can require inheritance to use class
- Can require derived class to override one or more methods

Pilot.java

```java
public abstract class Pilot {
  private Flight currentFlight;
  public void fly(Flight f) {
    if(canAccept(f))
      currentFlight = f;
    else
      handleCantAccept();
  }
  public abstract
  private void handleCantAccept() { System.out.println("Can't accept"); }
}
```

Overriding Abstract Methods

CargoOnlyPilot.java

```
public class
 CargoOnlyPilot extends Pilot {
  @Override
  public boolean
   canAccept(Flight f) {
    return f.getPassengers() == 0;
  }
}
```

FullLicensePilot.java

```
public class
 FullLicensePilot extends Pilot {
  @Override
  public boolean
   canAccept(Flight f) {
    return true;
  }
}
```

Inheritance and Constructors

Constructors are not inherited

- Each class has its own constructors

Derived Class Constructors

Constructing a derived class instance

A base class constructor is always called

By default no-argument version called

Can explicitly call a base constructor

Use super keyword

Must be first line of constructor

Flight.java

```
public class Flight {
  private int flightNumber;
  public Flight() {  }
  public Flight(int flightNumber) {
    this.flightNumber = flightNumber;
  }
} // other members elided
```

Constructors and Inheritance

Main.java

```java
Flight f175 = new Flight(175);

CargoFlight cf =
          new CargoFlight();

CargoFlight cf294 =
          new CargoFlight(294);
```

$\times$

CargoFlight.java

```java
class CargoFlight extends Flight {

  // has no explicit constructors

}
```

CargoFlight.java

```
public class CargoFlight extends Flight {
  float maxCargoSpace = 1000.0f;
  public CargoFlight(int flightNumber) {
    super(flightNumber);
  }
  public CargoFlight(int flightNumber, float maxCargoSpace) {
    super(flightNumber);
    this.maxCargoSpace = maxCargoSpace;
  }
```

CargoFlight.java

```java
public class CargoFlight extends Flight {
  float maxCargoSpace = 1000.0f;
  public CargoFlight(int flightNumber) {
    super(flightNumber);
  }
  public CargoFlight(int flightNumber, float maxCargoSpace) {
     this(flightNumber);
    this.maxCargoSpace = maxCargoSpace;
  }
```

CargoFlight.java

```java
  public CargoFlight() { }

  public CargoFlight(float maxCargoSpace) {

    this.maxCargoSpace = maxCargoSpace;

  }

} // other members elided
```

Constructors and Inheritance

Main.java

```
CargoFlight cf294 =
    new CargoFlight(294);

CargoFlight cf85 =
    new CargoFlight(85, 2000.0f);
```

CargoFlight.java

```
                Flight(flightNumber)

CargoFlight(int flightNumber)

CargoFlight(int flightNumber,
            float maxCargoSpace)

CargoFlight()

CargoFlight(float maxCargoSpace)
```

Constructors and Inheritance

Main.java

```
CargoFlight cf294 =
    new CargoFlight(294);

CargoFlight cf85 =
    new CargoFlight(85, 2000.0f);

CargoFlight cf =
    new CargoFlight();
```

CargoFlight.java

```
CargoFlight(int flightNumber)

CargoFlight(int flightNumber,
            float maxCargoSpace)
```

$$\boxed{\texttt{CargoFlight()}} \longrightarrow \boxed{\texttt{Flight()}}$$

```
CargoFlight(float maxCargoSpace)
```

Constructors and Inheritance

Main.java

```
CargoFlight cf294 =
    new CargoFlight(294);

CargoFlight cf85 =
    new CargoFlight(85, 2000.0f);

CargoFlight cf =
    new CargoFlight();

CargoFlight cfBig =
    new CargoFlight(5000.0f);
```

CargoFlight.java

```
CargoFlight(int flightNumber)

CargoFlight(int flightNumber,
            float maxCargoSpace)

CargoFlight()

CargoFlight(float maxCargoSpace)
```

$\xrightarrow{}$ `Flight()`

Summary

super reference

- Refers to the current object
- Treats as instance of base class

Summary

Preventing inheritance

- Mark class as final

Preventing method overriding

- Mark method as final

Requiring inheritance

- Mark class as abstract

Requiring method overriding

- Mark method as abstract

Summary

Constructors are not inherited

- Each class has its own constructors

Constructing a derived class instance

- A base class constructor always called
- By default calls no-argument version
- Can explicitly call specific constructor