Class Inheritance

Jim Wilson

@hedgehogjim | blog.jwhh.com | jimw@jwhh.com

What to Expect in This Module

- Inheritance basics
- Member hiding and overriding
- The Object class
- Object equality
- The super keyword
- Final and abstract
- Inheritance and constructors

Class Inheritance

A class can be declared to inherit (a.k.a. derive) from another class

Use the "extends" keyword

Derived class has characteristics of base class

Can add specialization

Class Inheritance

```
CargoFlight cf = new CargoFlight();

cf.add1Package(1.0, 2.5, 3.0);

Passenger jane = new Passenger(0,2);
cf.add1Passenger(jane);
```

```
public class CargoFlight extends Flight {
  float maxCargoSpace = 1000.0f;
  float usedCargoSpace;

  public void add1Package(float h, float w, float d) {
    double size = h * w * d;
    if(hasCargoSpace(size))
      usedCargoSpace += size;
    else
      handleNoSpace();
  }

  private boolean hasCargoSpace(float size) {
    return usedCargoSpace + size <= maxCargoSpace;
  }

  private
    Syste
  }
}
```

```
public class Flight {
  // other members elided for clarity

  public void add1Pasenger(Passenger p) { ... }
}
```

Class Inheritance

A class can be declared to inherit (a.k.a. derive) from another class

Use the "extends" keyword

Derived class has characteristics of base class

Can add specialization

Can be assigned to base class typed references

Class Inheritance

```
Flight f = new CargoFlight();

Passenger jane = new Passenger(0,2);
f.add1Passenger(jane);

f.add1Package(1.0, 2.5, 1.5);
```

```
Flight[] squadron = new Flight[5];
squadron[0] = new Flight();
squadron[1] = new CargoFlight();
squadron[2] = new CargoFlight();
squadron[3] = new Flight();
squadron[4] = new CargoFlight();
```

```
public class CargoFlight extends Flight {
  float maxCargoSpace = 1000.0f;
  float usedCargoSpace;

  public void add1Package(float h, float w, float d) {
    double size = h * w * d;
    if(hasCargoSpace(size))
      usedCargoSpace += size;
    else
      handleNoSpace();
  }

  private boolean hasCargoSpace(float size) {
    return usedCargoSpace + size <= maxCargoSpace;
  }

  private void handleNoSpace() {
    System.out.println("Not enough space");
  }

}
```

Class Inheritance

A class can be declared to inherit (a.k.a. derive) from another class

Use the "extends" keyword

Derived class has characteristics of base class

Can add specialization

Fields hide base class fields with same name

Can be assigned to base class typed references

Class Inheritance

```
Flight f1 = new Flight();
System.out.println(f1.seats);

CargoFlight cf = new CargoFlight;
System.out.println(cf.seats);

Flight f2 = new CargoFlight();
System.out.println(f2.seats);

f2.add1Passenger();
cf.add1Passenger();
```

150

12

150

```
public class CargoFlight extends Flight {
  // other members elided for clarity
  int seats = 12;
}
```

```
public class Flight {
  // other members elided for clarity
  int seats = 150;

  public void add1Passenger() {
    if(hasSeating())
      passengers += 1;
    else
      handleTooMany();
  }

  private boolean hasSeating() {
    return passengers < seats;
  }
}
```

Class Inheritance

A class can be declared to inherit (a.k.a. derive) from another class

Use the "extends" keyword

Derived class has characteristics of base class

Can add specialization

- Fields hide base class fields with same name
- Can be assigned to base class typed references
- Methods override base class methods with same signature

Class Inheritance

```
Flight f1 = new Flight();
System.out.println(f1.getSeats());

CargoFlight cf = new CargoFlight;
System.out.println(cf.getSeats());

Flight f2 = new CargoFlight();
System.out.println(f2.getSeats());

f2.add1Passenger();
cf.add1Passenger();
```

150

12

150

```
public class CargoFlight extends Flight {
  // other members elided for clarity
  @Override
  int getSeats() { return 12; }
}
```

```
public class Flight {
  // other members elided for clarity
  int getSeats() { return 150; }

  public void add1Passenger() {
    if(hasSeating())
      passengers += 1;
    else
      handleTooMany();
  }

  private boolean hasSeating() {
    return passengers < getSeats();
  }
}
```

Object Class

The Object class is the root of the Java class hierarchy

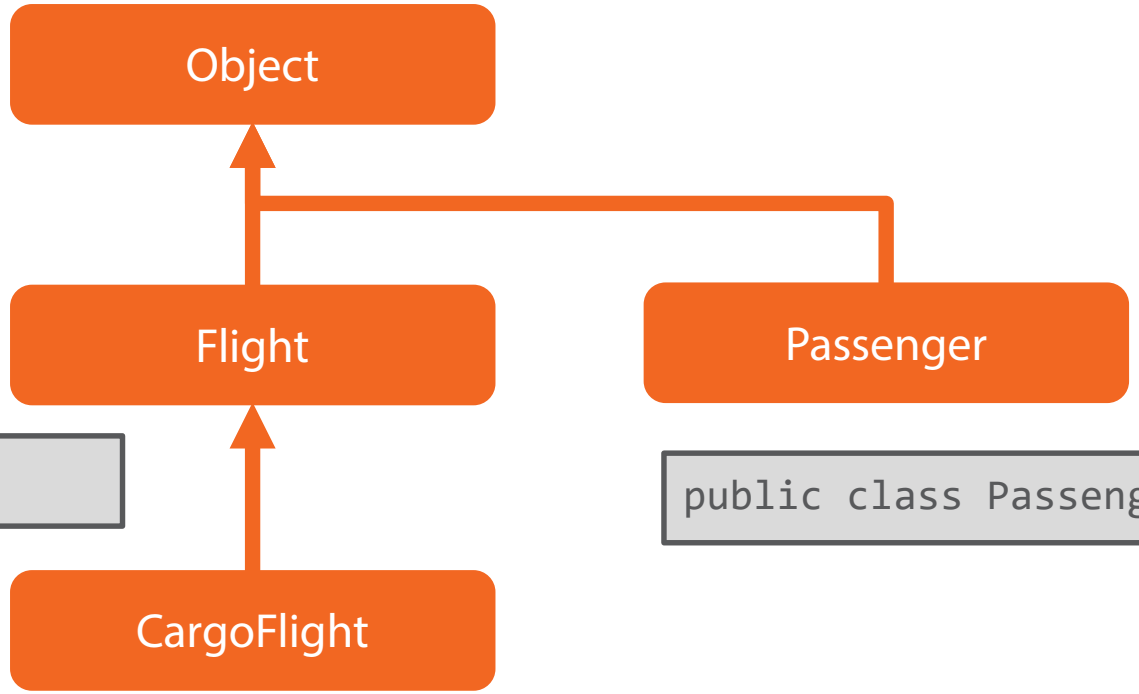
Every class has the characteristics of the Object class

Useful for declaring variables, fields and parameters that can reference any class or array instance

Defines a number of methods that are inherited by all objects

Inheriting from Object

Every class inherits directly or indirectly from the Object class




```
public class Flight {extends} Object
```

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

```
public class CargoFlight extends Flight { ... }
```

Object References

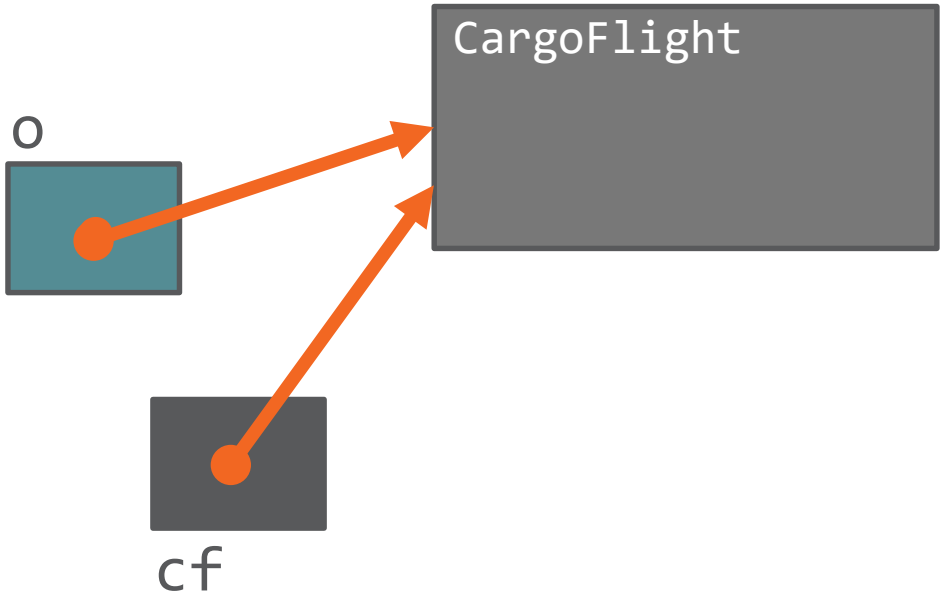
```
Object[] stuff = new Object[3];
stuff[0] = new Flight();
stuff[1] = new Passenger(0, 2);
stuff[2] = new CargoFlight();
```

```
Object o = new Passenger();

o = new Flight[5];

o = new CargoFlight();
o.add1Package(1.0, 2.5, 3.0);
```

Object References




```
Object o = new Passenger();

o = new Flight[5];

o = new CargoFlight();
o.add1Package(1.0, 2.5, 3.0);

if(o instanceof CargoFlight) {
   CargoFlight cf = (CargoFlight) o;
   cf.add1Package(1.0, 2.5, 3.0);
}
```

Object Class Methods

Method	Description
clone	Create a new object instance that duplicates the current instance
hashCode	Get a hash code for the current instance
getClass	Return type information for the current instance
finalize	Handle special resource cleanup scenarios
toString	Return string of characters representing the current instance
equals	Compare another object to the current instance for equality

Equality

What does it mean to be equal?

It depends!

```
Flight f1 = new Flight(175);
Flight f2 = new Flight(175);

if(f1 == f2)
  // do something

if(f1.equals(f2))
  // do something

Passenger p = new Passenger();
if(f1.equals(p))
  // do something
```

FALSE

TRUE

FALSE

```
class Flight {
  // other members elided for clarity
  private int flightNumber;
  private char flightClass;

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

    Flight other = (Flight) o;
    return
      flightNumber == other.flightNumber &&
      flightClass == other.flightClass;
  }
}
```

Special Reference: super

- Similar to *this*, *super* is an implicit reference to the current object
  - *super* treats the object as if it is an instance of its base class
  - Useful for accessing base class members that have been overridden

```
Flight f1 = new Flight(175);
Flight f2 = f1;

if(f1.equals(f2))
  // do something
```

```
class Flight {
  // other members elided for clarity
  private int flightNumber;
  private char flightClass;
  @Override
  public boolean equals(Object o) {
    if(!(o instanceof Flight))
      return false;

    Flight other = (Flight) o;
    return
      flightNumber == other.flightNumber &&
      flightClass == other.flightClass;
  }
}
```

```
  if(super.equals(o))
    return true;
```

Controlling Inheritance and Overriding

By default all classes can be extended
and derived classes have the option to use or override inherited methods

A class can change these defaults

Use final to prevent
inheriting and/or overriding

Using Final

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

```
public class CargoFlight extends Flight
  // other members elided for clarity

  public final void add1Package(float h, float w, float d) {
    double size = h * w * d;
    if(hasCargoSpace(size))
      usedCargoSpace += size;
    else
      handleNoSpace();
  }

  private boolean hasCargoSpace(float size) {
    return usedCargoSpace + size <= maxCargoSpace;
  }

  private void handleNoSpace() {
    System.out.println("Not enough space");
  }

}
```

Controlling Inheritance and Overriding

By default all classes can be extended
and derived classes have the option to use or override inherited methods

A class can change these defaults

Use final to prevent
inheriting and/or overriding

Use abstract to require
inheriting and/or overriding

Using Abstract

```
public abstract class Pilot {
  private Flight currentFlight;

  public void fly(Flight f) {
    if(canAccept(f))
      currentFlight = f;
    else
      handleCantAccept();
  }

  public abstract boolean canAccept(Flight f);

  private void handleCantAccept() {
    System.out.println("Can't accept");
  }
}
```

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

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

Inheritance and Constructors

Constructors are not inherited

A base class constructor must always be called

By default, base class' no-argument constructor is called

Can explicitly call a base class constructor using *super* followed by parameter list

Must be first line of constructor

Inheritance and Constructors

```
public class Flight {
  // other members elided for clarity
  private int flightNumber;

  public Flight() { }

  public Flight(int flightNumber) {
    this.flightNumber = flightNumber;
  }
}
```

```
public class CargoFlight extends Flight {
  // other members elided for clarity
  float maxCargoSpace = 1000.0f;

  public CargoFlight(int flightNumber) {
    super(flightNumber);
  }

  public CargoFlight(int flightNumber,
                     float maxCargoSpace) {
    super(flightNumber);
    this.maxCargoSpace = maxCargoSpace;
  }

  public CargoFlight() { }

  public CargoFlight(float maxCargoSpace) {
    this.maxCargoSpace = maxCargoSpace;
  }
}
```

```
Flight f175 = new Flight(175);
CargoFlight cf= new CargoFlight();
CargoFlight cf294 = new CargoFlight(294);
CargoFlight cf85 = new CargoFlight(85, 2000.0f);
CargoFlight cfBig = new CargoFlight(5000.0f);
```

Demo
CalcEngine with Specialized Classes

Summary

- Inheritance allows a new class to be defined with the characteristics of another
  - Use the extend keyword
- Derived class can override base class methods
  - Optionally use @Override annotation
- All classes derive from Object class either directly or indirectly
- By default, object references are only equal when referencing the same instance
  - Can override Object.equals to provide new behavior
- super accesses current object as if instance of base class
- final and abstract provide control over class inheritance and method overriding
- Constructors are not inherited