Class Initializers and Constructors

Jim Wilson

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

What to Expect in This Module

- Establishing initial state
- Field Initializers
- Constructors
- Constructor chaining & visibility
- Initialization blocks
- Initialization and construction order

Establishing Initial State

When an object is created, it is expected to be in a useful state

Often the default state established by Java is not enough

The object may need to set values or execute code

Mechanisms for Establishing Initial State

Java provides 3 mechanisms for establishing initial state

- Field initializers
- Constructors
- Initialization blocks

Field Initial State

A field's initial state is established as part of object construction

Fields receive a "zero" value by default

byte short int long	float double	char	boolean	Reference types
0	0.0	'\u0000'	false	null

You don't have to accept the default value

Field Initializers

- Allow you to specify a field’s initial value as part of its declaration
  - Can be a simple assignment
  - Can be an equation
  - Can reference other fields
  - Can be a method call

```
public class Earth {
  long circumferenceInMiles = 24901;
  long circumferenceInKilometers =
     Math.round(circumferenceInMiles * 1.6d);
}
```

Constructor

- Executable code used during object creation to set the initial state
  - Have no return type
  - Every class has at least one constructor

```
public class Flight  {
  private int passengers;
  private int seats;

  public Flight() {
     seats = 150;
     passengers = 0;
  }

  // other members elided for clarity
}
```

Constructor

- Executable code used during object creation to set the initial state
  - Have no return type
  - Every class has at least one constructor
    - If no explicit constructors, Java provides one
  - A class can have multiple constructors
    - Each with a different parameter list

```
Passenger bob = new Passenger();
bob.setCheckedBags(3);

Passenger jane = new Passenger(2);
jane.setCheckedBags(3);
```

```
public class Passenger {
  private int checkedBags;
  private int freeBags;
  // accessors & mutators elided for clarity

  private double perBagFee;

  public Passenger() { }

  }
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }


}
```

Chaining Constructors

- One constructor can call another
  - Use the this keyword followed by parameter list
  - Must be the first line

```
Passenger jane = new Passenger(2);
jane.setCheckedBags(3);

Passenger jane = new Passenger(2, 3);
```

```
public class Passenger {
  // fields & methods elided for clarity
  public Passenger() {

  }
  public Passenger(int freeBags) {
     this.freeBags = freeBags;
  }

  public Passenger(int freeBags, int checkedBags) {
     this(freeBags); freeBags;
     this.checkedBags = checkedBags;
  }


}
```

Constructor Visibility

- Use access modifiers to control constructor visibility
  - Limits what code can perform specific creations

```
Passenger cheapJoe = new Passenger(0.01d);

Passenger fred = new Passenger(2);

Passenger jane = new Passenger(2, 3);
```

```
public class Passenger {
  // fields & methods elided for clarity
  public Passenger() {

  }
  public Passenger(int freeBags) {
    this(freeBags > 1 ? 25.0d : 50.0d);
    this.freeBags = freeBags;
  }

  public Passenger(int freeBags, int checkedBags) {
    this(freeBags);
    this.checkedBags = checkedBags;
  }

  public Passenger(double perBagFee) {
  private Passenger(double perBagFee) {
    this.perBagFee = perBagFee;
  }
}
```

Demo

CalcEngine with Field Initializers and Constructors

Initialization Blocks

- Initialization blocks shared across all constructors
  - Executed as if the code were placed at the start of each constructor

```
public class Flight {
  private int passengers, flightNumber, seats ;= 150;
  private char flightClass;

  public Flight() {
     seats = 150;
     passengers = 0;

  }

  public Flight(int flightNumber) {

     this.flightNumber = flightNumber;
  }

  public Flight(char flightClass) {

     this.flightClass = flightClass;
  }
}
```

Initialization Blocks

- Initialization blocks shared across all constructors
  - Executed as if the code were placed at the start of each constructor
  - Enclose statements in brackets outside of any method or constructor

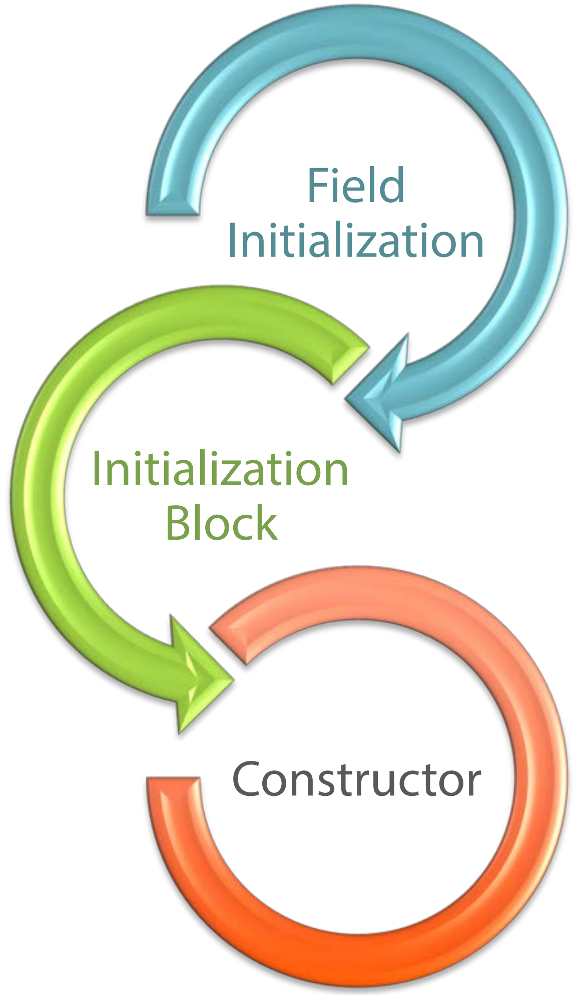
```
public class Flight {
  private int passengers, flightNumber, seats = 150;
  private char flightClass;
  private boolean[] isSeatAvailable;

  public Flight() {
     isSeatAvailable = new boolean[seats];

     for(int i = 0; i < seats; i++)
        isSeatAvailable[i] = true;
  }


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

  public Flight(char flightClass) {
     this();
     this.flightClass = flightClass;
  }
}
```

Initialization Blocks

- Initialization blocks shared across all constructors
  - Executed as if the code were placed at the start of each constructor
  - Enclose statements in brackets outside of any method or constructor

```
public class Flight {
  private int passengers, flightNumber, seats = 150;
  private char flightClass;
  private boolean[] isSeatAvailable;

  public Flight() { }
  {
     isSeatAvailable = new boolean[seats];

     for(int i = 0; i < seats; i++)

        isSeatAvailable[i] = true;
  }


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

  public Flight(char flightClass) {
     this();
     this.flightClass = flightClass;
  }
}
```

Initialization and Construction Order



```
public class OverInitializedClass {
  private int theField = 1;
  public int getTheField() { return theField ; }

  {
     theField = 2;
  }

   public OverInitializedClass() {
      theField = 3;
   }
 }
```

```
OverInitializedClass c =
        new OverInitializedClass();

System.out.println(c.getTheField();
```

3

Summary

- Objects should be created in some useful state
- Field initializers provide an initial value as part of the declaration
- Every class has at least one constructor
  - If no explicit constructor, Java provides one with no arguments
  - You can provide multiple constructors with differing argument lists
- One constructor can call another
  - Call must be first line
- Initialization blocks share code across constructors
- Keep the initialization and construction order in mind