Implementing Class Constructors and Initializers

Jim Wilson
Mobile Solutions Developer & Architect

@hedgehogjim | jwhh.com

Overview

Default initial state of fields

Field initializers

Constructors

Chaining constructors

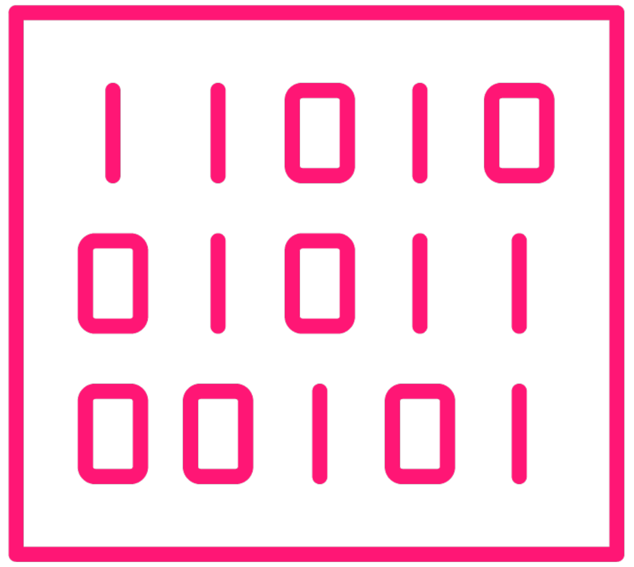
Constructor visibility

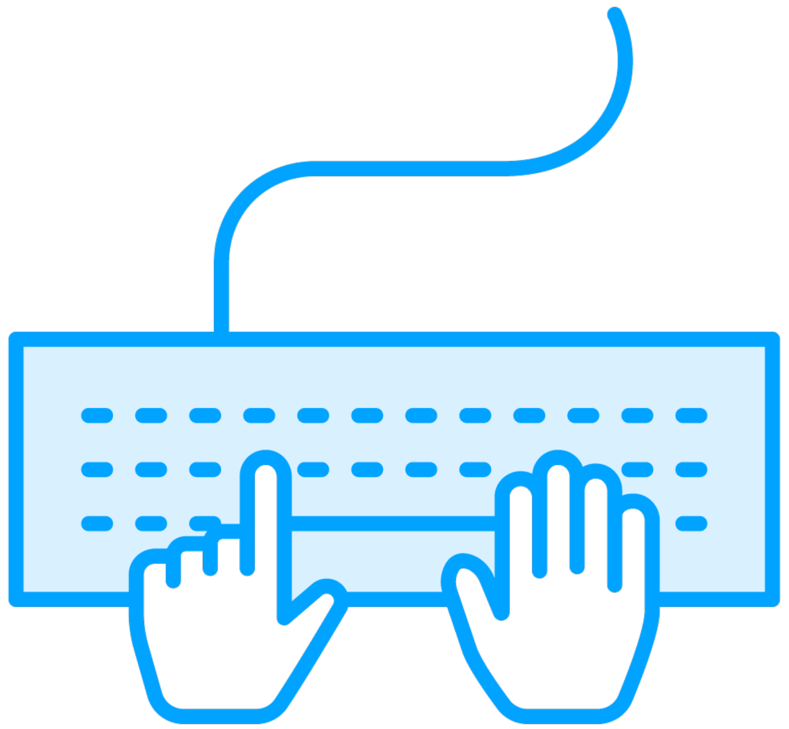
Initialization blocks

Initialization and construction order

Class Initial State

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





Default initial state set by Java often not enough

May need specific action

Set field values

Execute code

Default Initial State of Fields

byte short int long	float double
0	0.0

Establishing Initial State

Three ways to establish initial state

Field initializers

Constructors

Initialization blocks

```java
public class Earth {
  long circumferenceInMiles;
  long circumferenceInKms = (long)(24901 * 1.6d);
}
```

Field Initializers

Specify field’s initial value as part of the field’s declaration

- Can be an equation
- Can include other fields

```java
public class Earth {
  long circumferenceInMiles = 24901;
  long circumferenceInKms = (long)(circumferenceInMiles * 1.6d);
}
```

Field Initializers

Specify field's initial value as part of the field's declaration

- Can be an equation
- Can include other fields
- Can include method calls

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

Field Initializers

Specify field’s initial value as part of the field’s declaration

- Can be an equation
- Can include other fields
- Can include method calls

Constructors

Code that runs during object creation

- Named same as the class
- No return type

Constructors

```
class Flight {
  private int passengers;
  private int seats;
  Flight() {
    seats = 150;
    passengers = 0;
  }
  // other members elided for clarity
}
```

Constructors

```
class Flight {
  private int passengers;
  private int seats = 150;
  Flight() {



  }
  // other members elided for clarity
}
```

Number of Constructors

Must have at least one

When no explicit constructor, Java provides one

Number of Constructors

Main.java

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

Passenger.java

```java
public class Passenger{
  private int checkedBags;
  private int freeBags;
  // getters and setters elided
  private double perBagFee;
  public Passenger() { }
}
```

Number of Constructors

Must have at least one

When no explicit constructor,
Java provides one

Can have multiple

Each must have a unique parameter list

Different number of parameters

Different parameter types

Number of Constructors

Main.java

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

Passenger nia = new Passenger(2);
```

Passenger.java

```
public class Passenger{
  // other members elided
  public Passenger() { }
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }
}
```

Number of Constructors

Main.java

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

Passenger nia = new Passenger(2);
```

Passenger.java

```java
public class Passenger{
  // other members elided
  public Passenger() { }
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }
}
```

Number of Constructors

Main.java

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

Passenger nia = new Passenger(2);
```

Passenger.java

```java
public class Passenger{
  // other members elided
  public Passenger() { }
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }
}
```

Chaining Constructors

One constructor can call another

- Must be first line of the constructor
- Use the this keyword followed by the parameter list

Chaining Constructors

```java
public class Passenger{
  // other members elided
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }
  public Passenger(int freeBags, int checkedBags) {
    this.freeBags = freeBags;
    this.checkedBags = checkedBags;
  }
}
```

Chaining Constructors

```
public class Passenger{
  // other members elided
  public Passenger(int freeBags) {
    this.freeBags = freeBags;
  }
  public Passenger(int freeBags, int checkedBags) {
    this(freeBags);
    this.checkedBags = checkedBags;
  }
}
```

Chaining Constructors

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

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

Chaining Constructors

```
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) {
  this.perBagFee = perBagFee;
}
```

Constructor Visibility

Constructors can be non-public

- Limits which code can perform specific types of instance creation

Constructor Visibility

Main.java

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

Passenger.java

```
public Passenger()
public Passenger(int freeBags)
public Passenger
       (int freeBags,
        int checkedBags)
public Passenger
       (double perBagFee)
```

Constructor Visibility

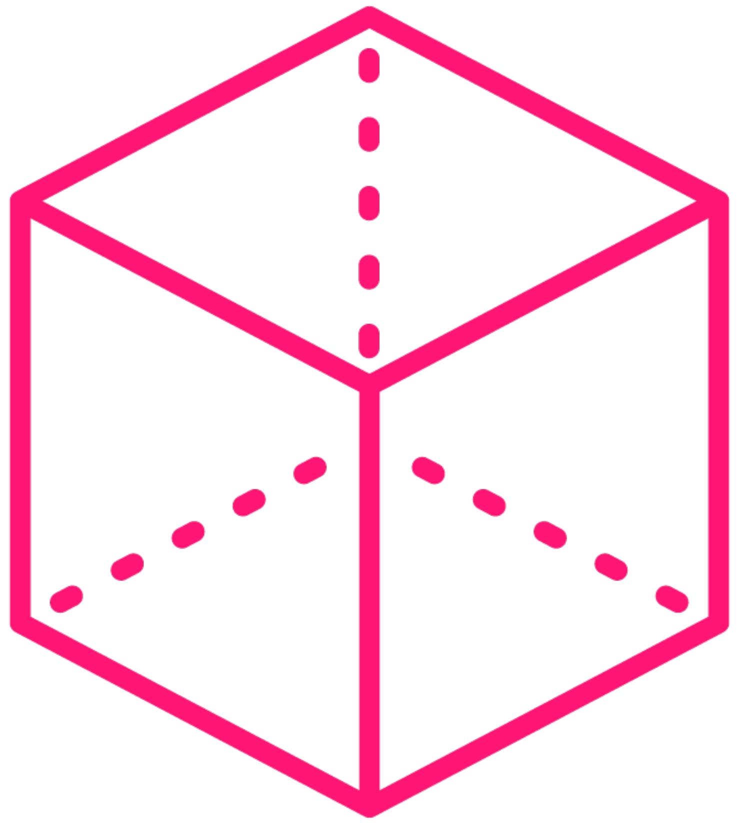
Main.java

```
Passenger cheapJoe =
            new Passenger(0.01d);
Passenger geetha =
            new Passenger(2);
Passenger santiago =
            new Passenger(2, 3);
```

Passenger.java

```
public Passenger()
public Passenger(int freeBags)
public Passenger
       (int freeBags,
        int checkedBags)
private Passenger
       (double perBagFee)
```

Initialization Blocks



Share code across all constructors

- Cannot receive parameters
- Place code within brackets outside of any method or constructor

A class can have multiple

- All always execute
- Execute in order starting at the top of the source file

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

  public Flight() {
    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;
  }
  // other members elided
}
```

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

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

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

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

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

Initialization and Construction Order

Field initializers

Initialization blocks

Constructors

Summary

Object initial state

- Initial state expected to be useful
- Java provides default field values

Field initializers

- Set initial value as part of declaration
- Can include an equation, other fields, and method calls

Summary

Constructors

- Code that runs during object creation
- Accept zero or more parameters
- Can have multiple

One constructor can call another

- Call must be first line of constructor
- Can pass parameters

Constructors can be non-public

- Limits which code can perform specific types of instance creation

Summary

Initialization blocks

- Code that runs during object creation
- Not tied to any specific constructor
- Cannot receive parameters