# A Closer Look at Parameters

Jim Wilson

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

# What to Expect in This Module

Parameter immutability

Constructor & method overloading

Variable number of parameters

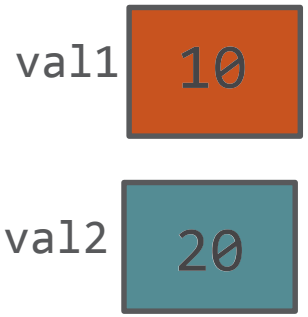
# Parameter Immutability

Parameters are passed by making a copy of the value

Known as passing "by-value"

Changes made to passed value
are not visible outside of method

# Parameter Immutability: Primitive Types



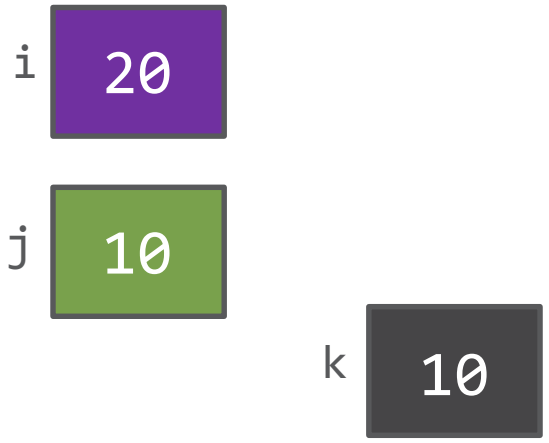

```
int val1 = 10;
int val2 = 20;
// print val1 & val2

swap(val1, val2);

// print val1 & val2
```

val1 →10
val2 →20

val1 →10
val2 →20




```
void swap(int i, int j) {
    int k = i;
    i = j;
    j = k;
    // print i & j
}
```

i →20
j →10

# Parameter Immutability: Classes

```
public class Flight {
  int flightNumber;
  // accessor & mutator elided for clarity

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

  // other members elided for clarity
}
```

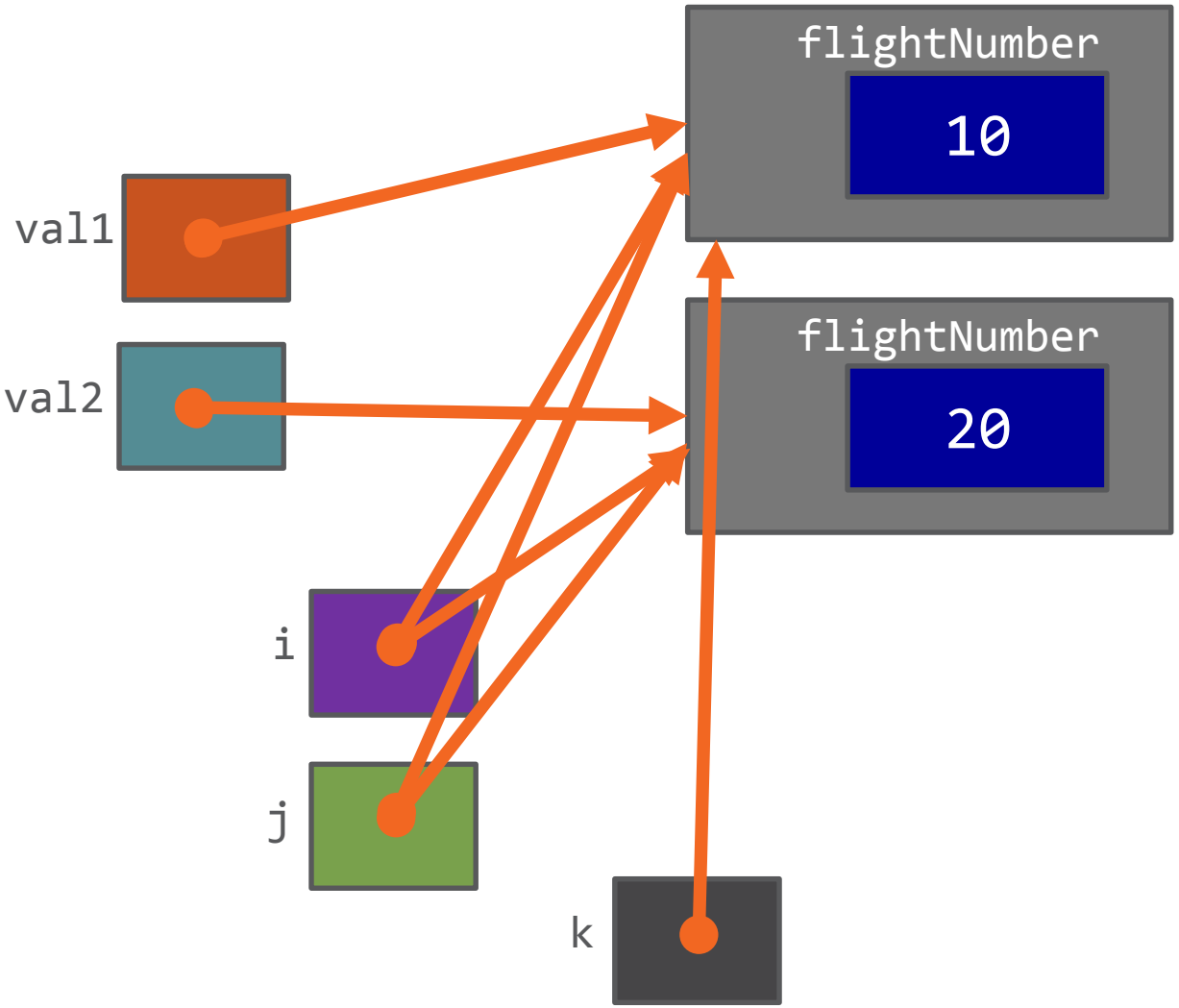
```
Flight val1 = new Flight(10);
```

```
void swap(Flight i, Flight j) {



}
```

# Parameter Immutability: Classes




```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
// print val1 & val2 flight #

swap(val1, val2);

// print val1 & val2 flight #
```

val1 →10
val2 →20

val1 →10
val2 →20

```
void swap(Flight i, Flight j) {
    Flight k = i;
    i = j;
    j = k;
    // print i & j flight #
}
```

i →20
j →10

# Parameter Immutability

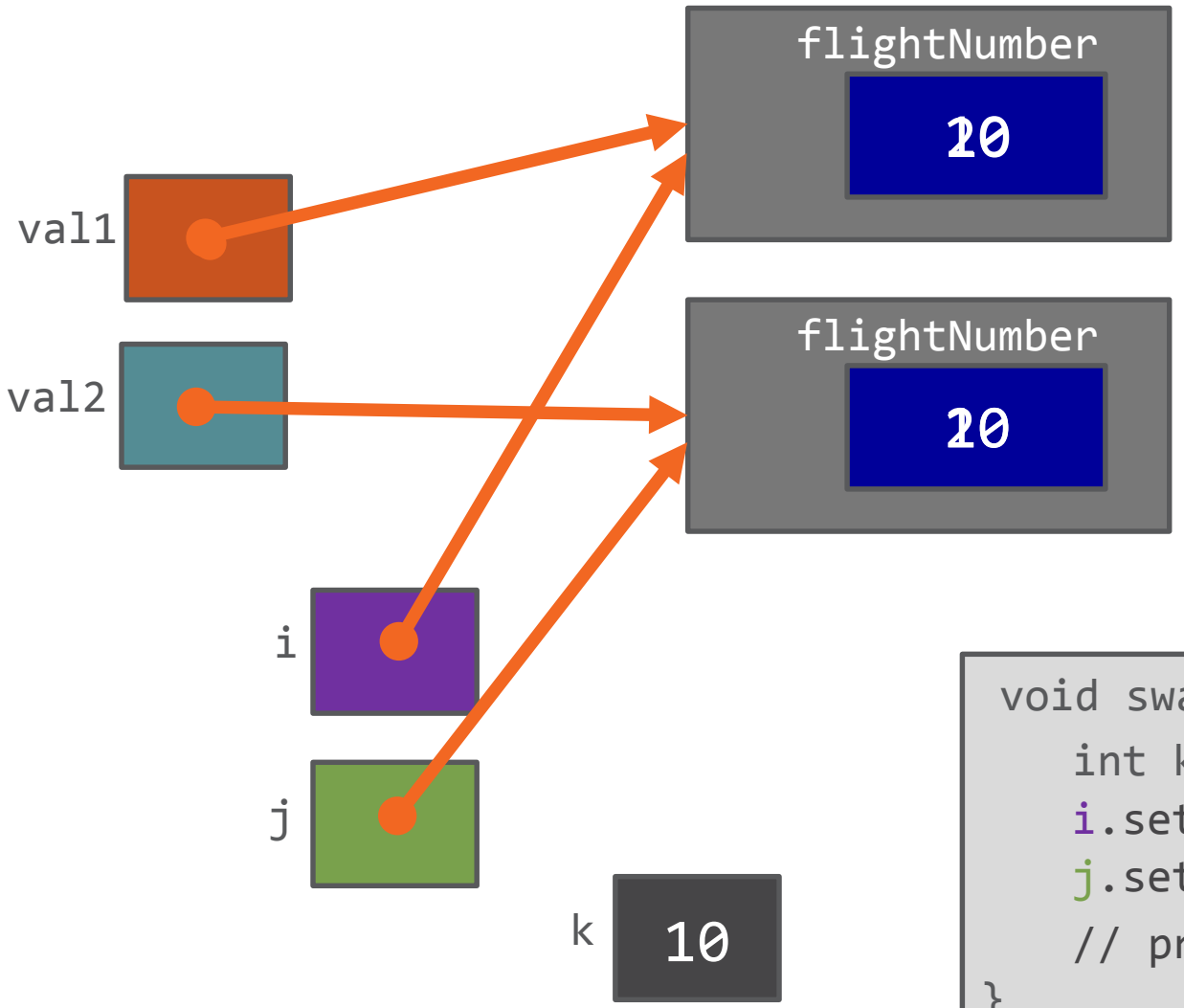
Parameters are passed by making a copy of the value

Known as passing "by-value"

Changes made to passed value are not visible outside of method

Changes made to members of passed class instances are visible outside of method

# Parameter Immutability: Class Members




```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
// print val1 & val2 flight #

swapNumbers(val1, val2);

// print val1 & val2 flight #
```

val1 →10
val2 →20

val1 →20
val2 →10

```
void swapNumbers(Flight i, Flight j) {
    int k = i.getFlightNumber();
    i.setFlightNumber(j.getFlightNumber());
    j.setFlightNumber(k);
    // print i & j flight #
}
```

i →20
j →10

# Overloading

A class may have multiple versions of its constructor or methods

Known as "overloading"

Each constructor and method must have a unique signature

Signature is made up of 3 parts

- Number of parameters

# Overloading

```
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);
     this.checkedBags = checkedBags;
  }

}
```

# Overloading

A class may have multiple versions of its constructor or methods

Known as "overloading"

Each constructor and method must have a unique signature

Signature is made up of 3 parts

- Number of parameters
- Type of each parameter

# Overloading

```
public class Flight {

  // fields & methods elided for clarity


  public Flight() {

  }


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


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

}
```

# Overloading

A class may have multiple versions of its constructor or methods

Known as "overloading"

Each constructor and method must have a unique signature

Signature is made up of 3 parts

- Name
- Number of parameters
- Type of each parameter

# Overloading

```
public class Flight {
  // other members elided for clarity
  int seats = 150, passengers;
  int totalCheckedBags;
  int maxCarryOns = seats * 2, totalCarryOns;

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

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

  private boolean hasCarryOnSpace(int carryOns) {
    return totalCarryOns + carryOns <= maxCarryOns;
  }
}
```

# Overloading

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

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

 public void add1Passenger(int bags) { ... }
   if(hasSeating()) {
     add1Passenger();
     totalCheckedBags += bags;
   }
 }
}
```

```
 public void add1Passenger(Passenger p) { ... }
   add1Passenger(p.getCheckedBags());
 }

 public void add1Passenger(int bags, int carryOns) { ... }
   if(hasSeating()&& hasCarryOnSpace(carryOns)) {
     add1Passenger(bags());
     totalCarryOns += carryOns;
   }
 }

 public void add1Passenger(Passenger p , int carryOns) {...}
   add1Passenger(p.getCheckedBags(), carryOns);
 }

}
```

# Overloading

```
Flight f = new Flight();
Passenger p1 = new Passenger(0,1);
Passenger p2 = new Passenger(0,2);

f.add1Passenger();
f.add1Passenger(2);
f.add1Passenger(p1);

short threeBags = 3;
f.add1Passenger(threeBags, 2);
f.add1Passenger(p2, 1);
```

```
public class Flight {
 // other members elided for clarity
 public void add1Passenger() { ... }

 public void add1Passenger(int bags) { ... }

 public void add1Passenger(Passenger p) { ... }

 public void add1Passenger(int bags, int carryOns) { ... }

 public void add1Passenger(Passenger p , int carryOns) { ... }
}
```

# Demo
## CalcEngine with Method Overloading

# Variable Number of Parameters

```
Flight f = new Flight();

Passenger janet = new Passenger(0,1);
Passenger john = new Passenger(0,2);

f.addPassengers(
    new Passenger[] { janet, john});

Passenger fred = new Passenger(0,2);
Passenger sarah = new Passenger(0,2);
Passenger susie = new Passenger(0,0);

f.addPassengers(
    new Passenger[] {fred, sarah, susie});
```

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

  public void addPassengers(Passenger[]  list) {
     if(hasSeating(list.length)) {
        passengers += list.length;
        for (Passenger passenger : list)
           totalCheckedBags +=
               passenger.getCheckedBags();
     }
     else
        handleTooMany();
  }

  private boolean hasSeating(int count) {
    return passengers + count <= seats;
  }
}
```

# Variable Number of Parameters

- A method can be declared to accept a varying number of parameter values
  - Place an ellipse after parameter type
  - Can only be the last parameter
  - Method receives values as an array

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

  public void addPassengers(Passenger[]. list) {
     if(hasSeating(list.length)) {
        passengers += list.length;
        for (Passenger passenger : list)
           totalCheckedBags +=
              passenger.getCheckedBags();
     }
     else
        handleTooMany();
  }

  private boolean hasSeating(int count) {
    return passengers + count <= seats;
  }
}
```

# Variable Number of Parameters

```
Flight f = new Flight();

Passenger janet = new Passenger(0,1);
Passenger john = new Passenger(0,2);

f.addPassengers(janet, john);
    new Passenger[] { janet, john});

Passenger fred = new Passenger(0,2);
Passenger sarah = new Passenger(0,2);
Passenger susie = new Passenger(0,0);

f.addPassengers(fred, sarah, susie);
    new Passenger[] {fred, sarah, susie});
```

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

  public void addPassengers(Passenger... list) {
     if(hasSeating(list.length)) {
        passengers += list.length;
        for (Passenger passenger : list)
           totalCheckedBags +=
               passenger.getCheckedBags();
     }
     else
        handleTooMany();
  }

  private boolean hasSeating(int count) {
    return passengers + count <= seats;
  }
}
```

# Summary

- Parameters are immutable
  - Changes made to passed value are not visible outside of method
  - Changes made to members of passed class instances are visible outside of method
- A class may have multiple versions of its constructor or methods
  - Each must have a unique signature
  - Signature includes name, number of parameters, type of each parameter
- A method can be declared to accept varying number of parameter values
  - Values received as an array
  - Must be last parameter