A Closer Look at Methods

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim jwhh.com

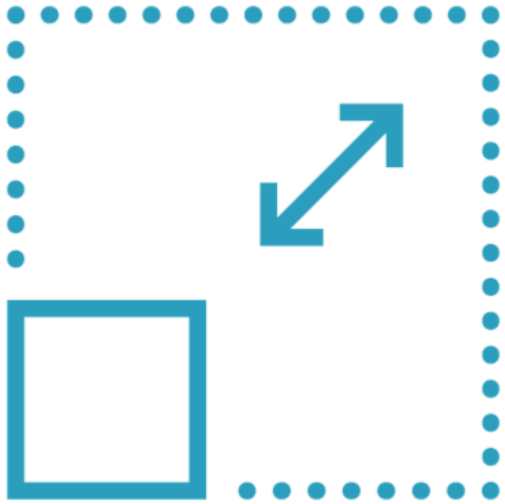
Overview

- Passing objects as parameters
- Effect of changes to object parameters
- Overloading
- Overloaded method resolution
- Object class and methods

Passing Objects as Parameters

Passed "by reference"

Parameter receives a copy of the reference



Changes to the reference

Visible within method

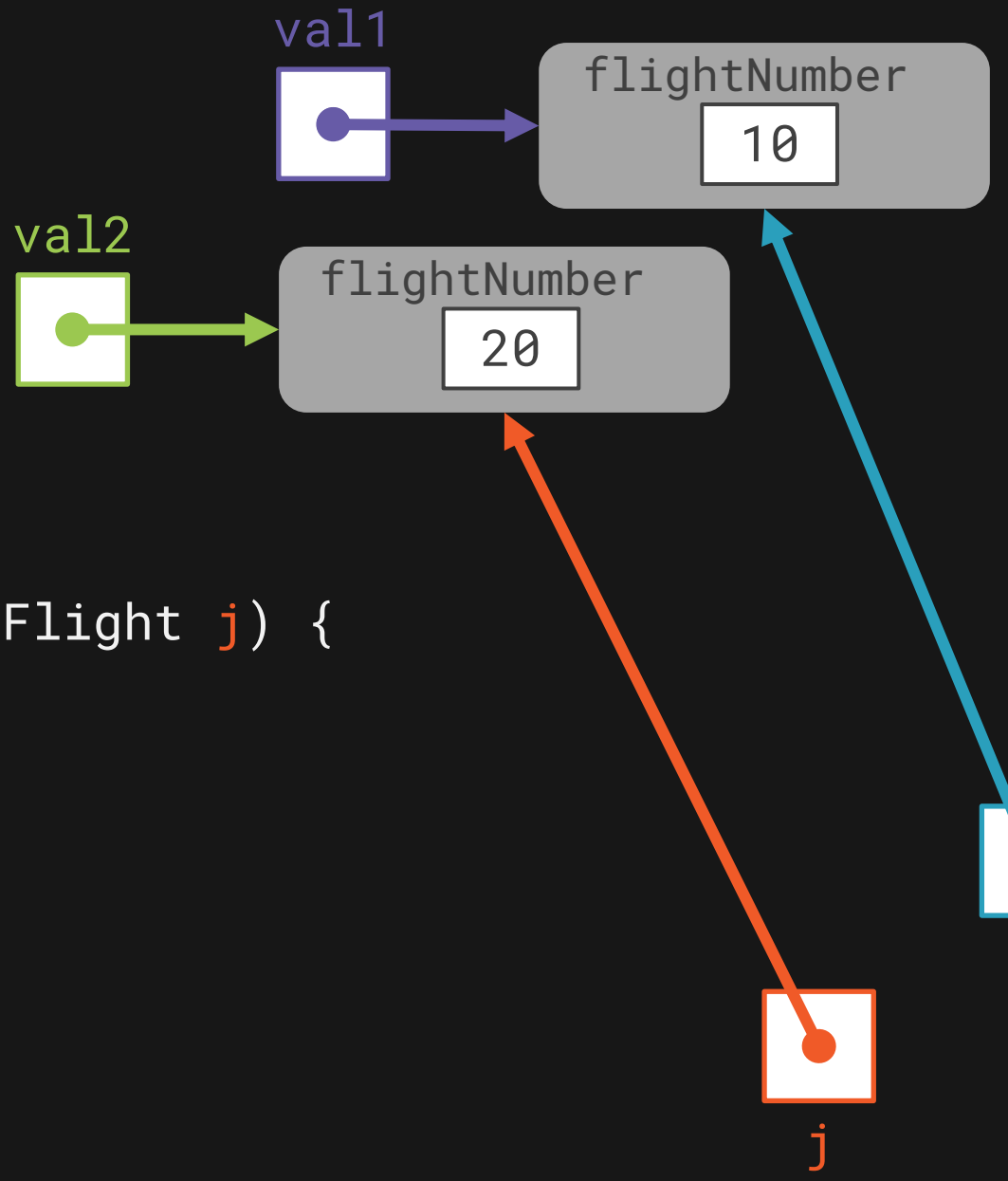
Not visible outside method

Flight.java

```
public class Flight {
  private int flightNumber;

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

Main.java

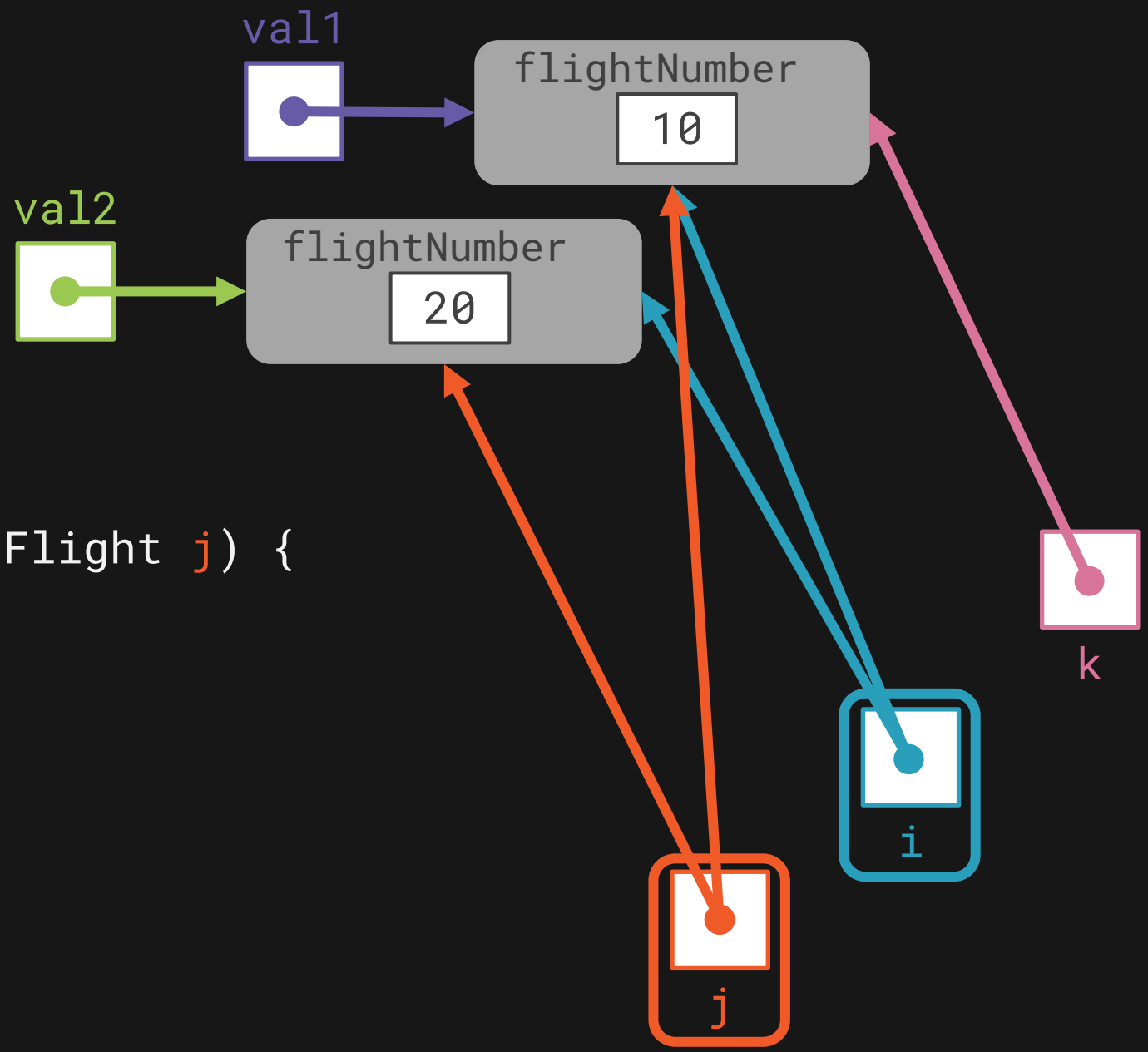
```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
swapFlight(val1, val2);
```




```
static void swapFlight(Flight i, Flight j) {
  Flight k = i;
  i = j;
  j = k;
}
```

Main.java

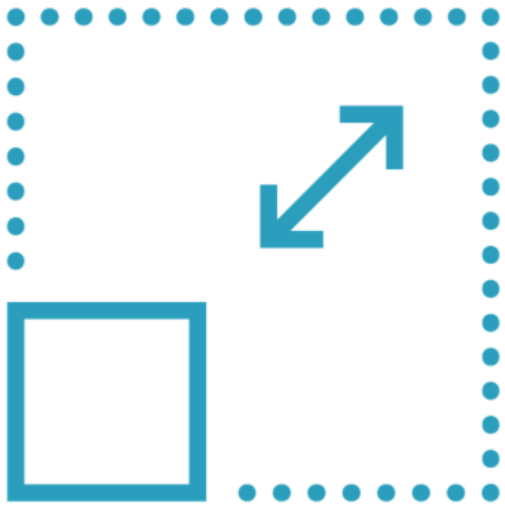
```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
swapFlight(val1, val2);
// print flight #'s

static void swapFlight(Flight i, Flight j) {
  Flight k = i;
  i = j;
  j = k
}
```



Passing Objects as Parameters

Passed "by reference"

Parameter receives a copy of the reference



Changes to the reference

Visible within method

Not visible outside method

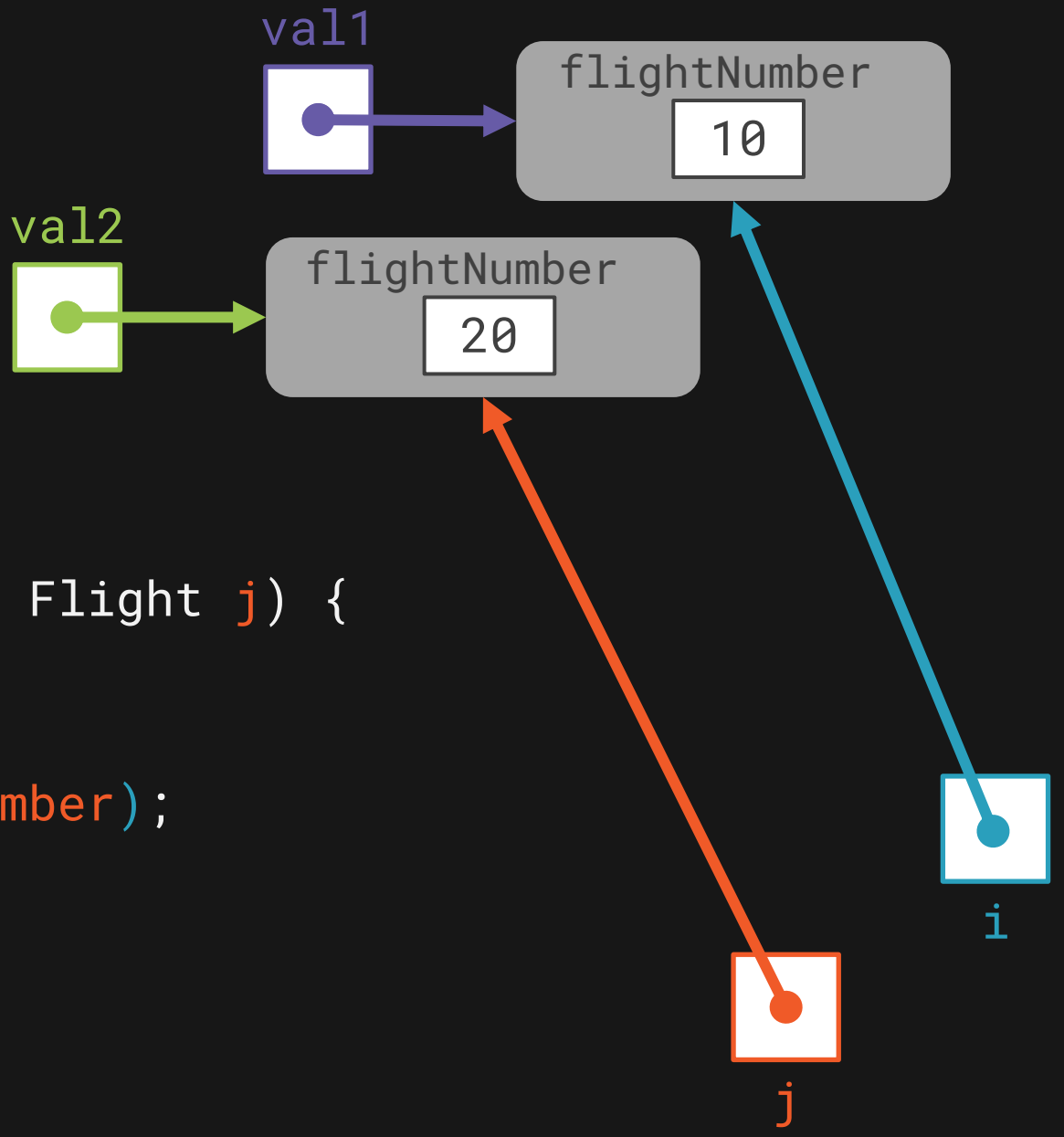
Changes to members

Visible within method

Visible outside method

Main.java

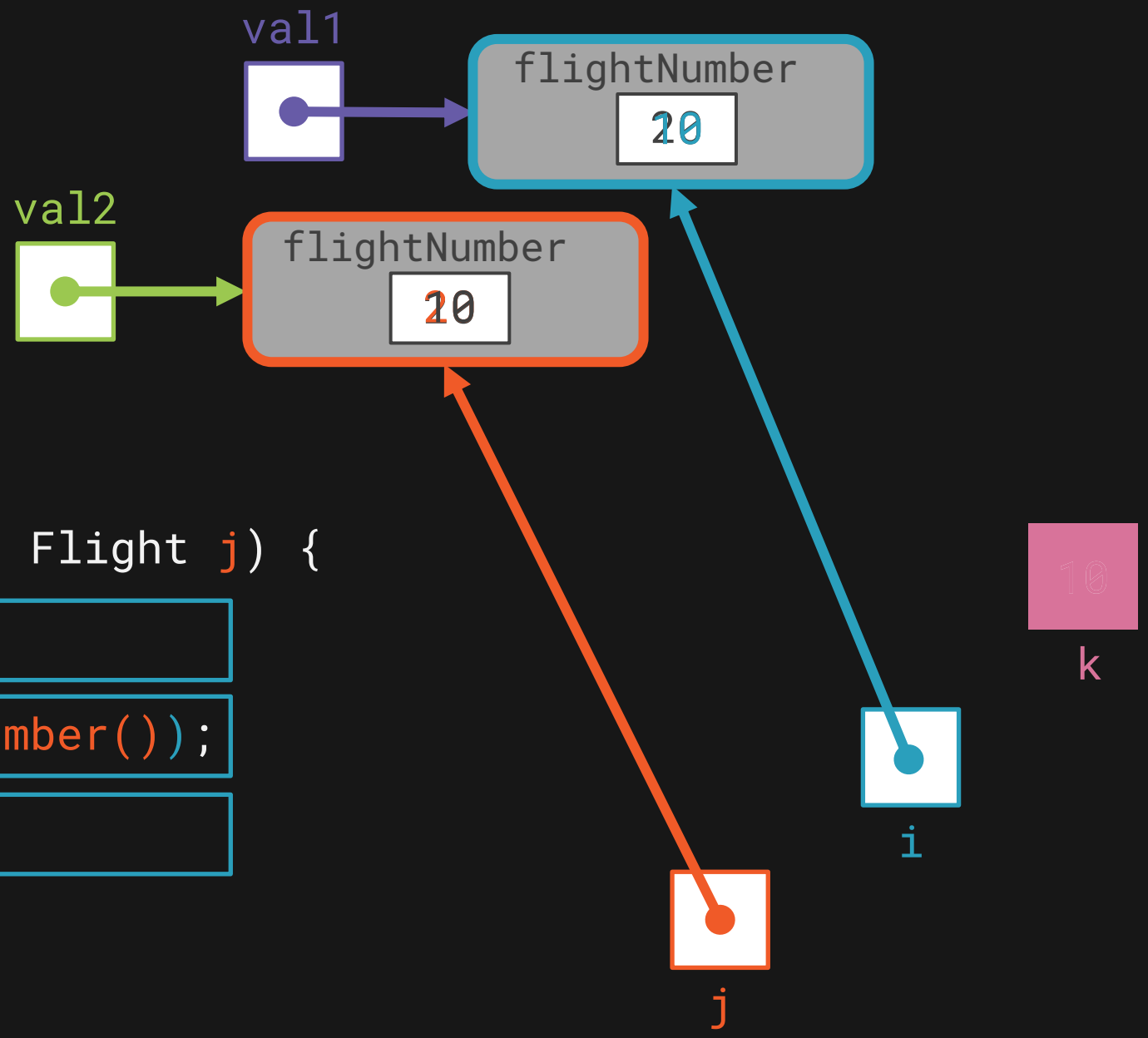
```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
swapNumbers(val1, val2);
```




```
static void swapNumbers(Flight i, Flight j) {
  int k = i.getFlightNumber();
  i.setFlightNumber(j.getFlightNumber);
  j.setFlightNumber(k);
}
```

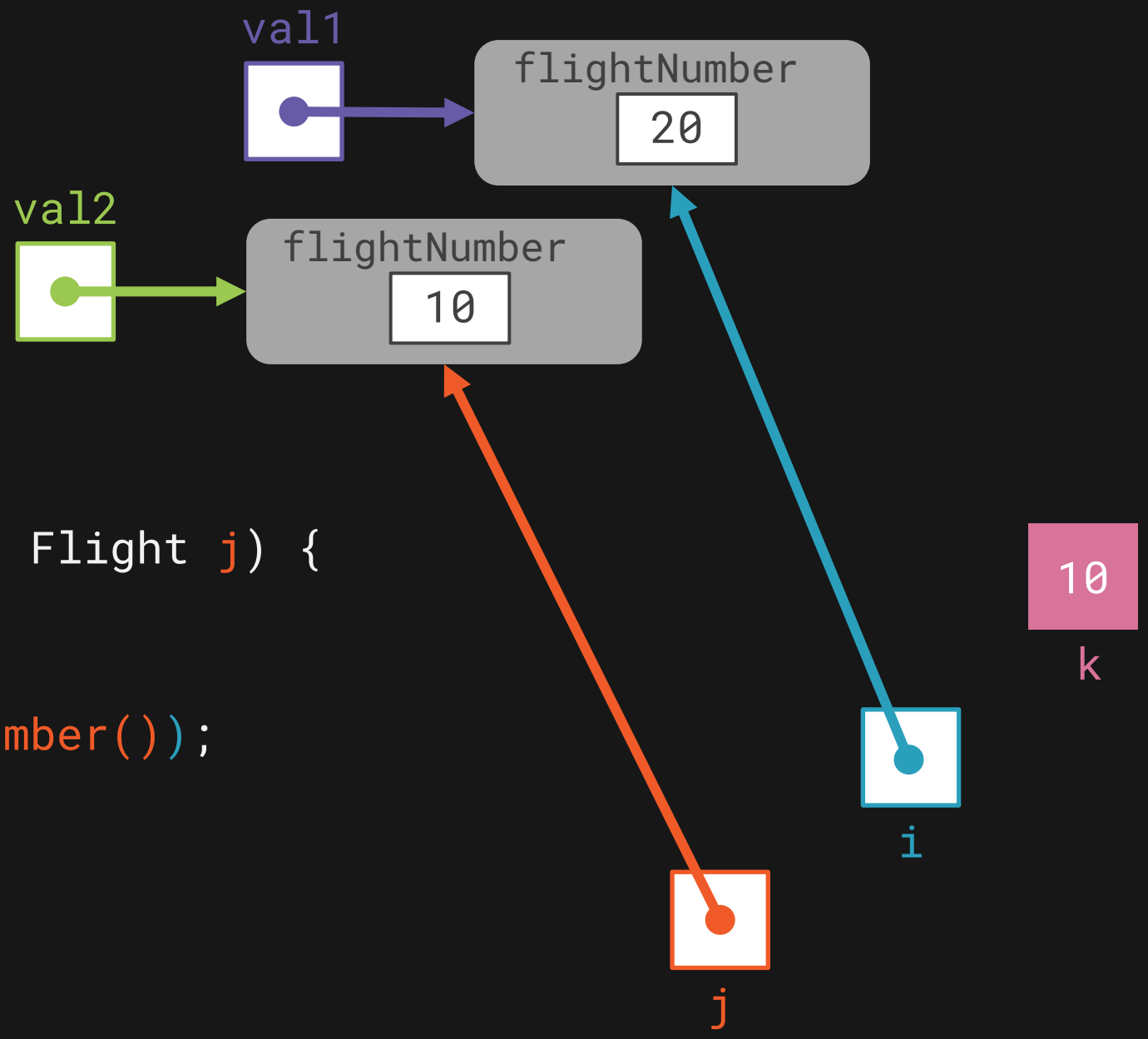
Main.java

```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
swapNumbers(val1, val2);


static void swapNumbers(Flight i, Flight j) {
    int k = i.getFlightNumber();
    i.setFlightNumber(j.getFlightNumber());
    j.setFlightNumber(k);
}
```



Main.java

```
Flight val1 = new Flight(10);
Flight val2 = new Flight(20);
swapNumbers(val1, val2);
// print flight #'s

static void swapNumbers(Flight i, Flight j) {
  int k = i.getFlightNumber();
  i.setFlightNumber(j.getFlightNumber());
  j.setFlightNumber(k);
}
```



Overloading

- Multiple versions of a method or constructor within a class

Overloading

```
class Passenger {
  Passenger() { . . . }
  Passenger(int freeBags) { . . . }
  Passenger(double perBagFee) { . . . }
  Passenger(int freeBags, int checkedBags) { . . . }

  // other members elided
}
```

Overloading

Each constructor and method must have a unique signature

Number of parameters

Overloading

```
class Passenger {
  Passenger() { . . . }
  Passenger(int freeBags) { . . . }
  Passenger(double perBagFee) { . . . }
  Passenger(int freeBags, int checkedBags) { . . . }

  // other members elided
}
```

Overloading

Each constructor and method must have a unique signature

Number of parameters

Type of each parameter

Overloading

```
class Passenger {
  Passenger() { . . . }
  Passenger(int freeBags) { . . . }
  Passenger(double perBagFee) { . . . }
  Passenger(int freeBags, int checkedBags) { . . . }

  // other members elided
}
```

Overloading

Each constructor and method must have a unique signature

Number of parameters

Type of each parameter

Method name

Flight.java

```
class Flight {
  int passengers, seats = 150;
  public void add1Passenger() {
    if(passengers < seats)
      passengers += 1;
  }
  private boolean hasSeating() {
    return passengers < seats;
  }
  // other members elided
}
```

Flight.java

```
class Flight {
  int passengers, seats = 150;
  public void add1Passenger() {
    if(hasSeating())
      passengers += 1;
  }
  private boolean hasSeating() {
    return passengers < seats;
  }
  // other members elided
}
```

Flight.java

```
class Flight {
  int passengers, seats = 150, totalCheckedBags;
  public void add1Passenger() {
    if(hasSeating())
      passengers += 1;
  }
  private boolean hasSeating() {
    return passengers < seats;
  }
  // other members elided
}
```

Flight.java

```
public void add1Passenger() {
  if(hasSeating())
    passengers += 1;
}
public void add1Passenger(int bags) {
  if(hasSeating()) {
    add1Passenger();
    totalCheckedBags += bags;
  }
}
```

Flight.java

```
public void add1Passenger(Passenger p) {
  add1Passenger(p.getCheckedBags());
}
public void add1Passenger(int bags, int carryOns) {
  if(carryOns <= 2)
    add1Passenger(bags);
}
public void add1Passenger(Passenger p, int carryOns) {
  add1Passenger(p.getCheckedBags(), carryOns);
}
```

Overloading

Main.java

```
Flight f = new Flight();

f.add1Passenger();

f.add1Passenger(2);

Passenger p1 =
  new Passenger(0, 1);

f.add1Passenger(p1);
```

Flight.java

```
add1Passenger()

add1Passenger(int bags)

add1Passenger(Passenger p)

add1Passenger(int bags,
              int carryOns)

add1Passenger(Passenger p,
              int carryOns)
```

Overloading

Main.java

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

f.add1Passenger(p2, 1);

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

Flight.java

```
add1Passenger()
add1Passenger(int bags)
add1Passenger(Passenger p)
add1Passenger(int bags,
              int carryOns)
add1Passenger(Passenger p,
              int carryOns)
```

Overloading

Main.java

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

f.add1Passenger(p2, 1);

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

Flight.java

```
add1Passenger()
add1Passenger(int bags)
add1Passenger(Passenger p)
add1Passenger(int bags,
              int carryOns)
add1Passenger(Passenger p,
              int carryOns)
```

Java supports inheritance

- Allows one class to be declared with characteristics of another

Object Class

Root of the Java class hierarchy

- An Object reference can reference an instance of any class
- Every class has characteristics of Object

Object References

Main.java

```
Object[] stuff = new Object[3];
stuff[0] = new Flight(123);
stuff[1] = new MathEquation();
stuff[2] = “I Like Java”;
```

Main.java

```
Object o = “Just a string”;
o = new Flight(456);
```

Object References

Main.java

```
Flight f = new Flight(123);
doWork(f);

Passenger p = new Passenger();
doWork(p);
```

Main.java

```
void doWork(Object o) {
  // do something with Object
  //   characteristics of o
}
```

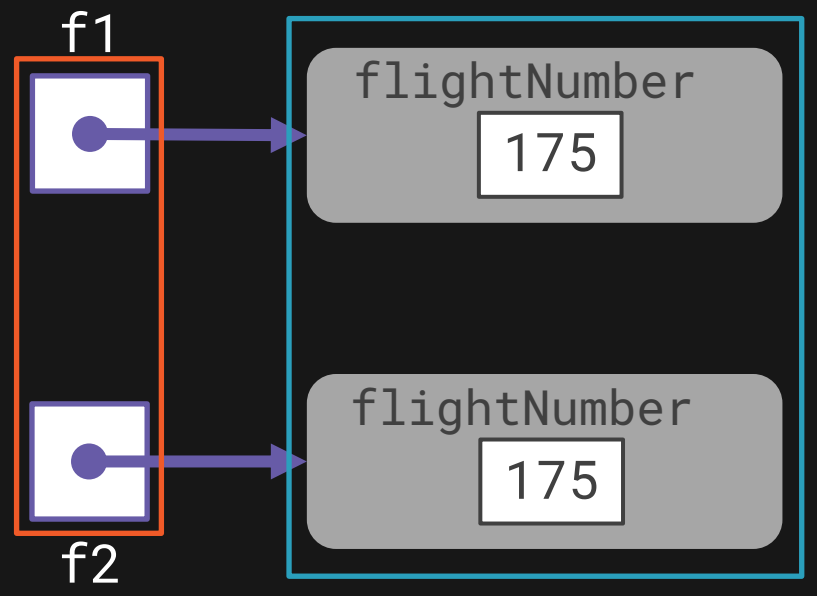
Object Class Methods

Method	Description

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
```



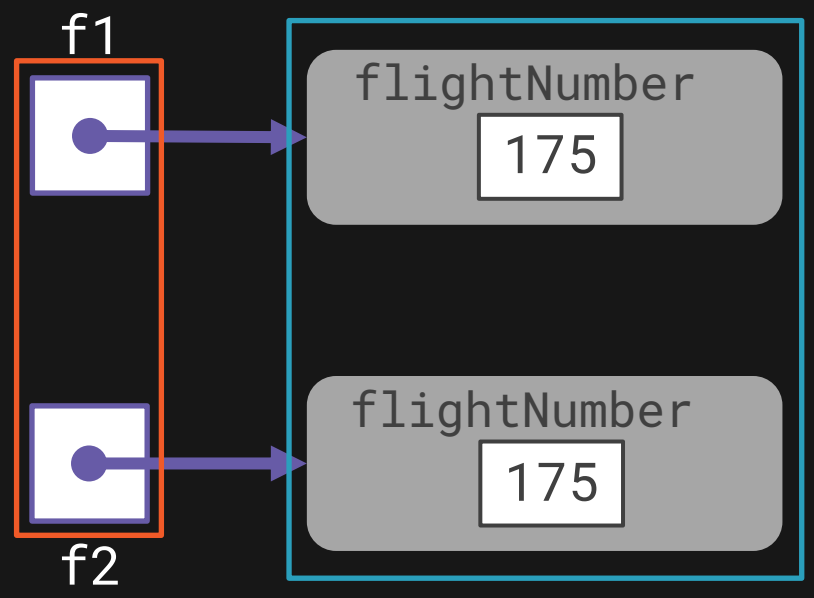
Flight.java

```
public class Flight {
  private int flightNumber;

  public boolean equals(Object o) {
    Flight flight = (Flight) o;
    return flightNumber == flight.flightNumber;
  }
} // other members elided
```

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
```



Flight.java

```
public class Flight {
  private int flightNumber;

  public boolean equals(Object o) {
    if (  o instanceof Flight )
        return false;
    Flight flight = (Flight) o;
    return flightNumber == flight.flightNumber;
  }
} // other members elided
```

Equality

What does it mean to be equal? ... It depends.

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

if(f1 == f2) // false
    // do something

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

Passenger p = new Passenger();

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

Summary

Objects are passed by-reference

- Reference is copied to the method

Method changes to the reference

- Not visible outside of the method

Method changes to referenced object

- Remain visible outside of the method

Summary

Overloading

- Multiple versions of a method or constructor within a class
- Each must have a unique signature

Parts of the signature

- Method name
- Number of parameters
- Type of each parameter

Summary

Object class

- Root class of the Java class hierarchy
- Object reference can reference an instance of any class
- Every class has Object characteristics