Today …

• NOTE – Midterm: Thursday, March 4th
• Go over Quiz 4
• More on constructors
• Exercise 6
• Lab 3 (cont.)
• Readings:
  – Ch. 8 pages 323-338
  – Ch. 9 pages 361-384
Quiz 4

1. How would you implement the constructor for B so that x is set to anX and y is set to aY?
   ```java
   public B(int anX, int aY) {
     super(anX);
     y = aY;
   }
   ```

2. Assume we want the getX() method in B to return “x+y”. How would you implement this method to return such a value?
   ```java
   public int getX() { return super.getX() + getY(); }
   ```

3. What are the return values for the statements below? Mark those that result in compile errors.
   ```java
   A a = new A(5);   B b = new B(6, 10);
   a. C.add(a, a);   10 // returns A’s getX() + A’s getX()
   b. C.add(a, b);   21 // returns A’s getX() + B’s getX()
   c. C.add(b, a);   21 // returns B’s getX() + A’s getX()
   d. C.add(b, b);   32 // returns B’s getX() + B’s getX()
   ```

More on constructors

Assume we have these classes:
```java
public class Animal {
  public Animal() {
    System.out.println("Making an Animal");
  }
}

public class Hippo extends Animal {
  public Hippo() {
    System.out.println("Making a Hippo");
  }
}

public class TestHippo {
  public static void main(String[] args) {
    System.out.println("Starting...");
    Hippo h = new Hippo();
  }
}
```
More on constructors

Assume we have these classes:

```java
public class Animal {
    public Animal() {
        System.out.println("Making an Animal");
    }
}

public class Hippo extends Animal {
    public Hippo() {
        System.out.println("Making a Hippo");
    }
}
```

(1) Calling `new Hippo()` places the `Hippo()` constructor on the call stack

(2) `Hippo()` invokes the superclass constructor which pushes the `Animal()` constructor on the stack

(3) `Animal()` invokes the superclass constructor pushing `Object()` on the stack

(4) `Object()` completes, and its stack frame is popped off the stack

(5) `Animal()` completes, and its stack frame is popped off the stack

(6) `Hippo()` completes, and its stack frame is popped off the stack

The object is now fully "constructed"
More on constructors

How do you invoke a superclass constructor?

```java
public class Duck extends Animal {
    int size;
    public Duck(int aSize) {
        Animal();
        size = aSize;
    }
}
```

**NO! This is not legal!**

**Is this legal?**

**What should we do?**

```java
public class Duck extends Animal {
    int size;
    public Duck(int aSize) {
        super();
        size = aSize;
    }
}
```

**Much better!**

(Unless you are also trying to learn C++)
More on super()

But do we really need to invoke super() here?

- No. The compiler will do it for us in some cases.

- If we don’t provide a constructor...
  
  ```java
  public MyClass {
  }
  ...
  ```

  the compiler adds a default constructor
  
  ```java
  public MyClass() {
    super();
  }
  ```

- If we do provide a constructor, but not a super() call
  
  - The compiler puts one in each overloaded constructor for us
  - Unless it calls another overloaded constructor (coming soon)

More on super()

But what happens in this case?

```java
public class Animal {
  int size;
  public Animal(int aSize) {
    size = aSize;
  }
}

public class Goose extends Animal {
  public Goose() {
  }
}
```

Compile Error!

symbol : constructor Animal()
location: class Animal
```
public Goose() {
    Calling super() in here
doesn’t match a constructor in Animal!
}
More on super()

How to fix this: One option

```java
public class Animal {
    int size;
    public Animal(int aSize) {
        size = aSize;
    }
}

public class Goose extends Animal {
    public Goose() {
        super(35);  // Pass a size to the superclasses constructor
    }
}
```

More on super()

How to fix this: Another (probably) better option

```java
public class Animal {
    int size;
    public Animal(int aSize) {
        size = aSize;
    }
}

public class Goose extends Animal {
    public Goose(int aSize) {
        super(aSize);  // Take the size as an argument
                        // (geese probably differ in size)
    }
}
```
Calling overloaded constructors

Suppose we have this

```java
public class Mini extends Car {
    private Color color;
    private int size;
    public Mini() {
        super("Mini");
        color = Color.RED;
        size = 120;
    }
    public Mini(Color aColor) {
        super("Mini");
        color = aColor;
        size = 120;
    }
    public Mini(int aSize) {
        super("Mini");
        color = Color.RED;
        size = aSize;
    }
    public Mini(Color aColor, int aSize) {
        super("Mini");
        color = aColor;
        size = aSize;
    }
}
```

A lot of redundancy here

Use this() to call overloaded constructors

Instead, we can use this() to simplify the code

```java
public class Mini extends Car {
    private Color color;
    private int size;
    public Mini() {
        this(Color.RED, 120);
    }
    public Mini(Color aColor) {
        this(aColor, 120);
    }
    public Mini(int aSize) {
        this(Color.RED, aSize);
    }
    public Mini(Color aColor, int aSize) {
        super("Mini");
        color = aColor;
        size = aSize;
    }
}
```

These "call" the primary constructor using the this(...) keyword

The primary constructor (does all the work)
**Rules on using this() and super()**

- The this(...) or super(...) construct calls must occur first in the constructor

- Every constructor can have a call to this(...) or super(...) but never both!

*E.g., this is wrong:*

```java
public Mini(int aSize) {
    super("Mini");
    this(Color.RED, aSize);
}
```

---

**Exercise 6**
Exercise 6

Compiler errors:

- 4 … no super() constructor (compiler tries to fill in)
- 6 … no super(String, String) constructor
- 7 … cannot mix this() and super()
- 9 … no super(int, String) constructor

Lab 3