Today …

• Quiz

• More on objects and classes
  – Getting out of main
  – The Java Heap and Garbage Collection
  – Data and object passing

• Reading assignments
  – Core: Ch. 4: 122-143
Quick! Get out of main!

• In a “real” OO application
  – You need objects talking to other objects (calling each other’s methods)
  – Not a static main() method doing all the work

• The two uses of main:
  – To test your real class
  – To launch/start your Java application

Quick! Get out of main!

We can use main() for “unit testing”

```java
public class Movie {
    private String title;
    public Movie(String theTitle) {
        title = theTitle;
    }
    public void playMovie() {
        System.out.println("Playing: "+ title);
    }

    public static void main(String[] args) {
        // make sure we can create an object
        Movie m = new Movie("Lost in cubicle space");
        // make sure we can play a movie
        m.playMovie();
    }
}
```

The application cares about this stuff
This tests that Movie works independently of the application
Compiling Multiple Source Files

• Lets say we have these classes:

In Player.java:

```java
public class Player {
    ...
}
```

In Game.java:

```java
public class Game {
    private Player p1;
    private Player p2;
    private Player p3;
    ...
}
```

In GameLauncher.java:

```java
public class GameLauncher {
    public static void main(String[] args) {
        Game theGame = new Game();
        ...
    }
}
```

• Assuming these are all in the same directory …
• How do we compile and run this program?

One approach: `javac *.

• This will create three compiled classes:
  – Player.class, Game.class, and GameLauncher.class
• We can then call: `java GameLauncher`
Compiling Multiple Source Files

- Lets say we have these classes:

  In Player.java:
  ```java
  public class Player {
    ...
  }
  ```

  In Game.java:
  ```java
  public class Game {
    private Player p1;
    private Player p2;
    private Player p3;
    ...
  }
  ```

  In GameLauncher.java:
  ```java
  public class GameLauncher {
    public static void main(String[] args) {
      Game theGame = new Game();
      ...
    }
  }
  ```

Even simpler: `javac GameLauncher.java`
- Again three .class files are created (like magic!)
- `javac` has “built in” dependency analysis
- It will compile dependencies as needed

This will become more complicated with packages …
- But for now we aren’t using packages
- So it is very simple!
The Java Heap

• Each time an object is created in Java
  – It goes into an area of memory called “The Heap”
  – All objects live on the heap

• When you create an object
  – The JVM allocates memory space on the heap
  – Based on how much that particular object needs
  – E.g., an object with 15 instance variables usually needs more space than one with 2 instance variables

The Java Heap

• What if the object isn’t needed anymore?
  – And, you want to free up the space
  – The JVM takes care of it for you using “Garbage Collection”

• The garbage collector looks for unreachable objects
  – When it finds an object that is not referenced
  – It removes (deletes) it from the heap!
Life on the heap

1. Book b = new Book();

Life on the heap

1. Book b = new Book();
2. Book c = new Book();
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;

Two object references can refer to the exact same object!

is d == c true?

Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();

The obj. reference 'c' now refers to a different object

---

Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
5. d = c;
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
5. d = c;

This object is toast
Garbage-collector bait
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
5. d = c;
6. d = null;

What happens if we call d.getTitle()?
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
5. d = c;
6. d = null;
7. b = null;
Life on the heap

1. Book b = new Book();
2. Book c = new Book();
3. Book d = c;
4. c = new Book();
5. d = c;
6. d = null;
7. b = null;

Arrays of object (references)

- When we create an array:
  
  Book[] books = new Book[3];

- What happens on the heap?
Arrays of object (references)

• When we create an array:
  
  ```java
  Book[] books = new Book[3];
  ```

• What happens on the heap?

  What does `books[1]` equal?
  
  null !!!

  What should we do?

  What happens if we call `books[1].turnPage()`?
Arrays of object (references)

- When we create an array:
  
  Book[] books = new Book[3];
  books[0] = new Book();
  books[1] = new Book();
  books[2] = new Book();

- What happens on the heap?

Exercise 5
Exercise 5

```java
public class HeapQuiz {
    private int id = 0;
    public void setId(int theId) {
        id = theId;
    }
    public static void main(String[] args) {
        HeapQuiz[] h = new HeapQuiz[5];
        for(int x = 0; x < 3; x++) {
            h[x] = new HeapQuiz();
            h[x].setId(x);
        }
        h[3] = h[1];
        h[4] = h[1];
        h[3] = null;
        h[4] = h[0];
        h[0] = h[3];
        h[3] = h[2];
        h[2] = h[0];
    }
}
```

Objects vs. Classes

- **Classes (blueprints)** tell the JVM how to create objects
  - The class “sets” the structure of the object
- Every object of a class can have **different** values for **instance variables**
- Can every object have **different methods**?
  - Technically, every instance of a class has the same methods
  - But, the methods can behave differently based on the values of instance variables
You can send things to methods

As we’ve seen, you can pass values into your methods

– If a method takes a parameter (formal argument)
– You must pass it a value (actual argument)

public class Dog {
    ...
    public void bark(int numOfBarks) {
        while(numOfBarks > 0) {
            System.out.println("ruff");
            numOfBarks = numOfBarks - 1;
        }
    }
}

You can get things back from methods

As we’ve seen, you can return values from methods

– If a method has a return type
– The method must return a value of that type

public class Dog {
    ...
    public int getSize() {
        return 60;
    }
    public void bark(int numOfBarks) {
        ...
    }
}

Dog d = new Dog();
D.bark(3);

Dog d = new Dog();
D.bark(3);
int s = D.getSize();
S = 60
You can send more than one thing to methods

Methods can have zero or more parameters

– Each is separated by commas
– Each has a type
– You have to pass the right number of arguments when calling the method

```
public class Dog {
    ...
    public void stay(int seconds, int treats) {
        ...
    }
    }
```

Java is “pass by value”

In other words “pass by copy”

– The variable value is copied to the parameter variable

```
public class Dog {
    ...
    public void bark(int numOfBarks) {
        ...
    }
    }
```
Java is “pass by value”

What is printed by this code?

```java
public class A {
    public void f(int x) {
        x = 5;
        System.out.println("x is " + x);
    }
}

int y = 4;
A a = new A();
a.f(y);
System.out.println("y is " + y);
```

```
x is 5
y is 4
```

Java is “pass by value”

Arguments can also have Classes as types

What is printed by this code?

```java
public class A {
    public void f(B b1) {
        b1.x = 5;
    }
}

public class B {
    public int x = 4; // note public
}

B b2 = new B();
A a = new A();
a.f(b2);
System.out.println("b2.x is " + b2.x);
```

```
b2.x is 5
```

Like y, b2 is not changed!
It still refers to the same object

```
y is never changed!
```

```
x = 5
4 (copy)
y = 4
```
Java is “pass by value”

Arguments can also have Classes as types

What is printed by this code?

```java
public class A {
    public void f(B b1) {
        b1.x = 5;
    }
}

public class B {
    public int x = 4; // note public
}

B b2 = new B();
A a = new A();
a.f(b2);
System.out.println("b2.x is " + b2.x);
```

b2.x is 5

b2’s object changed

b2’s reference didn’t change