Beyond the Single `main()` Method

- Many classes can be written using only one `main()` method call, containing all our code
- Many classes will have more than one method, however
- Some classes we write won’t have a `main()` method at all
- In fact, much code you have written is built this way...

You might only write a single `main()` method but:

1. Your code may use classes written by the instructor, like Oval, Rectangle, etc., that *don’t* have their own `main()` methods
2. Your code may use built-in classes, like `String`, `Scanner`, etc., that *also don’t* have their own `main()` methods

In each such case, you have *used* many other methods, like `setLocation()` or `length()`

Now it’s time to *create/write* your own methods to do things

Multiple Methods in a Class

- We can write as many methods in a class as we like
- Each will start with its own method declaration, and will contain a block of valid code
- Each must be complete before another method starts

```java
public class MethodRunner {
    public void sayHello() {
        System.out.println( "Hello!" );
    }
    private void sayGoodbye() {
        System.out.println( "Goodbye!" );
    }
}
```
private vs. public Methods

- Methods can be either **public** or **private**
- This **does not** affect compiling this code at all
- This **does not** affect what this code actually does at all
- Instead, it affects **where** we can legally **access** (i.e., **call**) it

```java
public class MethodRunner {
    public void sayHello() {
        System.out.println( "Hello!" );
    }
    private void sayGoodbye() {
        System.out.println( "Goodbye!" );
    }
}
```

A method with **public access**

A method with **private access**

public vs. private Method Access

- A **private** method can’t be called in any **other class** in the usual way:
  1. Instantiate object of class type
  2. Call method we wish to use

```java
public class MethodRunner {
    public void sayHello() {
        // code here
    }
    private void sayGoodbye() {
        // code here
    }
}
```

public class Foo {
    public static void main( String[] args ) {
        MethodRunner runner = new MethodRunner();
        runner.sayHello();
    }
}

Since `sayHello()` is public, we can call it out here in the separate `Foo` class

```java
public class MethodRunner {
    public void sayHello() {
        // code here
    }
    private void sayGoodbye() {
        // code here
    }
}
```

public class Bar {
    public static void main( String[] args ) {
        MethodRunner runner = new MethodRunner();
        runner.sayGoodbye();
    }
}

This code results in a **compile error**

private vs. public Method Access

- If we make a method in a class C **private**, it can only be called from within the class C itself
- If we make a method in C **public**, it can be called by:
  1. Any object instance from the class C itself
  2. **Any other** object that runs the `C()` constructor to create an object of type C, and then uses it to call the method

```java
public class MethodRunner {
    public void sayHello() {
        // code here
    }
    private void sayGoodbye() {
        // code here
    }
}
```

```java
public class Foo {
    public static void main( String[] args ) {
        MethodRunner runner = new MethodRunner();
        runner.sayHello();
    }
}
```

**sayGoodbye()** is **private**, and so we can’t call it here

This code results in a **compile error**
**public vs. private Method Access**

- Inside a single class, it doesn’t matter whether methods are public or private.
- Any other code can call every method in the same class.
- Since we are inside the same class, we don’t need an object to call the method.
- The compiler/JVM automatically looks for any methods we call this way inside the class itself.

```java
public class MethodRunner {
    public void sayHello() {
        // code here
    }
    private void sayGoodbye() {
        // code here
    }
    private void sayBoth() {
        sayHello();
        sayGoodbye();
    }
}
```

Here, we can directly call either of the other methods.

**Constructors**

- **Constructor method**
  - Used to create an object.
  - Has the same name as the class.
  - Has no return type – this is NOT the same as void.

```java
public class DrawingGizmo {
    <<constructor>>
    DrawingGizmo()

    <<update>>
    void moveForward()
    void turnClockwise()
    void turnCounterclockwise()
    void dontDraw()
    void draw()
    void delay2Sec()
}
```

**Using Constructors**

- How do you call a constructor?
  - You’ve already done so many times!
  - Use `new`.

```java
Oval o = new Oval();
Random rand = new Random();
Window win = new Window();
String s = new String();
Car Giulia = new Car();
```

These are calls to the constructors for the Oval, Random, Window, String, and Car classes.

**Multiple Constructors**

- Can have multiple constructors in a class:
  - Must have different parameter lists.
  - Recall the Team class from Program 2.
  - Java "knows" which one to use based on the parameter list you provide in the call.

```java
public class DrawingGizmo {
    <<constructor>>
    DrawingGizmo()
    DrawingGizmo(Color, Color)

    <<update>>
    void moveForward()
    void turnClockwise()
    void turnCounterclockwise()
    void dontDraw()
    void draw()
    void delay2Sec()
```
Multiple Constructors

Car Giulia = new Car();

Calls a constructor that takes no parameters.

Car MC40 = new Car("Mini", "Cooper S", 2004, 25000, 180, 190);

Calls a constructor that takes six parameters: String, String, int, int, int

This Week & Next

Meetings this week:
- Monday/Wednesday: regular classroom
- Tuesday: in the CS Lab (16 Wing)
- Friday: No class

Reading 05: Ch. 6 due Thursday, Oct 31 at Noon

Program 05: due 11:59 PM, Wednesday October 30

Office Hours: Wing 212
- Monday/Friday: 2:15 PM–3:15 PM
- Tuesday: 2:30 PM–1:30 PM
- Wednesday: 12:05 PM–1:00 PM