

CS 1331 Introduction to Object Oriented Programming

Review 1: Java Programs, Variables, Values, Control Structures, Arrays, Classes

Christopher Simpkins

`chris.simpkins@gatech.edu`

The Anatomy of a Java Program

```
import java.util.Random;

public class Hello {

    private static final String[] GREETINGS =
        {"Hello, world!", "Hi there!", "W'sup!"};

    private String greeting;

    private Hello() {
        Random rand = new Random();
        int greetingsIndex = rand.nextInt(GREETINGS.length);
        greeting = GREETINGS[greetingsIndex];
    }

    public Hello(int anIndex) {
        int greetingsIndex = anIndex % GREETINGS.length;
        greeting = GREETINGS[greetingsIndex];
    }

    public String getGreeting() {
        return greeting;
    }
}
```

The `import` Statement

The statement

```
import java.util.Random;
```

allows us to use the `java.util.Random` class in our program.
Without the import, the line

```
Random rand = new Random(1);
```

would produce the following error:

```
Hello.java:11: error: cannot find symbol
    Random rand = new Random(1);
                  ^
symbol:   class Random
location: class Hello
```

The `class` Declaration

A class is declared with the following syntax:

```
public class Hello { ... }
```

- `public` means that the class is visible to any object within the class's package, or within any Java file that imports the class
- `class` means we're declaring and defining a class
- the code between `{` and `}` define the class
- Every Java source file must contain exactly one public class. In this case `Hello` is public, so it must be in a file named `Hello.java`

static and final Variables

```
private static final String[] greetings =  
    {"Hello, world!", "Hi there!", "W'sup!"};
```

- `private` means the variable is only visible within instances of the `Hello` class
- `static` means that there is exactly one copy of this variable for all instances of the `Hello` class, even if no `Hello` objects have been instantiated
- `final` means we can only assign a value to this variable once. After that, its value is constant. Note that any variable can be `final`, not just `static` variables.
- `String[]` is the variable's type (in particular, a `String` array). Every variable has a type, which restricts the values it may be assigned.
- `greetings` is the variable's name
- In this case, we have both declared the variable `greetings` and initialized it with the value `{"Hello, world!", "Hi`

Instance Variables

```
private String greeting;
```

is an instance variable declaration.

- `private` means it is only visible within the `Hello` class
- `String` is the variables type
- `greeting` is the variable's name
- Every `Hello` object has it's own `greeting` instance variable
- Since we didn't initialize `greeting`, its value is `null` (until we execute a constructor)

No-arg Constructors

```
public Hello() {  
    Random rand = new Random(10);  
    int greetingsIndex = rand.nextInt(greetings.length);  
    greeting = greetings[greetingsIndex];  
}
```

- Constructors initialize an object of a class. In this case, the constructor initializes the `greeting` instance variable.
- Constructors are called with operator `new`, as in `Hello h = new Hello();`
- After `Hello h = new Hello();`, `h` holds the address of a `Hello` object which has some randomly assigned `greeting` instance variable.
- `h` is a *reference* to a `Hello` object

What does `Random rand = new Random(10);` do?

Constructors With Parameters

```
public Hello(int anIndex) {  
    int greetingsIndex = anIndex % greetings.length;  
    greeting = greetings[greetingsIndex];  
}
```

- To call this constructor, provide an argument in the call to `new`, as in `Hello h = new Hello(1);`
- Providing the argument in the `new` call selects a particular constructor, in this case the constructor that takes one `int` parameter

Getter Methods

```
public String getGreeting() {  
    return greeting;  
}
```

is a “getter” method, a.k.a., an “accessor”.

- `getGreeting()` returns the value of the `greeting` instance variable for the object on which it is invoked
- In general, the preferred way of naming getters is `getMyVariable`, where `myVariable` is an instance variable.
- For boolean instance variables, the naming convention is `isMyFlag`, where `myFlag` is a boolean instance variable.
- Using this naming convention makes a class usable as a Java Bean (just get in the naming habit for now)
- “Setters”, a.k.a. “mutators”, are named the same way, e.g., `setGreeting(String aNewGreeting)`
- Since we don’t have any setters in our `Hello` class, instances of `Hello` are immutable

The `main` Method

The method

```
public static void main(String[] args) {  
    Hello h = new Hello();  
    System.out.println(h.getGreeting());  
}
```

makes the `Hello` class executable.

- The signature must match `public static void main(String[] args)`
- It means that we can run the class with the `java` command

Compiling a Java Program

Given the definition of the `Hello` class, we can compile it like this:

```
$ javac Hello.java
```

- `$` is the command prompt.
- The argument to the `javac` command is the name of a Java source file ending in `.java`
- This example must be executed in the same directory `Hello.java` is located
- Executing the command above produces a `Hello.class` file (provided there are no compile errors)

Running a Java Program

Since `Hello` has a `main` method, we can run it:

```
$ java Hello
```

- The argument to the `java` command is the name of a compiled class
- Java finds this class on the classpath (which includes the current directory by default) and executes its `main` method
- This example must be executed in the same directory `Hello.class` is located

Maintaining Class Invariants

This `Card` class won't allow a client to set an invalid rank:

```
public class Card {
    private final String[] VALID_RANKS =
        {"2", "3", "4", "5", "6", "7", "8", "9",
         "10", "jack", "queen", "king", "ace"};
    private String rank;

    public void setRank(String rank) {
        if (!isValidRank(rank)) {
            throw new IllegalArgumentException("Invalid rank.");
        }
        this.rank = rank;
    }
    private boolean isValidRank(String someRank) {
        return Arrays.asList(VALID_RANKS).contains(someRank);
    }
}
```

- Types restrict the values you may set for a variable to a particular domain.
- With encapsulation you can further restrict the domain of

Static Variables

```
public class Doberman {  
    private static int dobieCount = 0;  
    private String name;  
  
    public Doberman(String name) {  
        this.name = name;  
        dobieCount++;  
    }  
    public String reportDobieCount() {  
        return name+" says there are "+dobieCount+" dobies.";  
    }  
}
```

What does this code print?

```
Doberman fido = new Doberman("Fido");  
System.out.println(fido.reportDobieCount());  
  
Doberman prince = new Doberman("Prince");  
System.out.println(prince.reportDobieCount());  
  
Doberman chloe = new Doberman("Chloe");  
System.out.println(chloe.reportDobieCount());
```

Review Questions: Variables and Values

- Is `int n = 2.2;` legal?
- What's the value of the expression `17 % 4`?
- What's the value of `n` after `int n = (int) 2.2;`?
- After the line above and `n++;`, what's the value of `n`?
- After the line above and `n += 2;`, what's the value of `n`?
- After the line above and `String s = "Answer: " + n;`, what's the value of `s`?

Given

```
boolean a = true;  
boolean b = false;
```

- What's the value of `x` after `boolean x = a || b;`?
- What's the value of `y` after `boolean y = a && b;`?

Review Questions: `if`-Statements

Will this code compile?

```
String condition = "true";  
if (condition) {  
    System.out.println("The true path.");  
} else {  
    System.out.println("The false path.");  
}
```

What will this code print?

```
boolean a = true;  
boolean b = false;  
if (a && b ) {  
    System.out.println("The true path.");  
} else {  
    System.out.println("The false path.");  
}
```


Short-Circuit Evaluation

What will this code print?

```
public class ShortCircuit {  
  
    private static int counter = 0;  
  
    private static boolean incrementCounter() {  
        counter++;  
        return true;  
    }  
  
    public static void main(String args[]) {  
        boolean a = true;  
        boolean b = false;  
        if (a || incrementCounter()) {  
            System.out.println("Evaluated (a || incrementCounter()).");  
        }  
        System.out.println("Counter = " + counter);  
        if (a && incrementCounter()) {  
            System.out.println("Evaluated (a && incrementCounter()).");  
        }  
        System.out.println("Counter = " + counter);  
    }  
}
```

Review Questions: Loops

How would you write this `while` loop as a `for` loop?

```
int n = 0;
while (n < 5) {
    System.out.println("Hip, hip, hooray!");
    n++;
}
```

Answer:

```
for (int n = 0; n < 5; n++) {
    System.out.println("Hip, hip, hooray!");
}
```

Equality

- To compare objects, like `Strings`, for value equality, use `equals`.
- For objects, `==` means identity equality – are two references aliases to the same object in memory.

What will this code print?

```
public class Foo {
    private String bar;
    public Foo(String bar) {
        this.bar = bar;
    }
    public boolean equals(Object other) {
        return this.bar.equals(((Foo) other).bar);
    }
    public static void main(String[] args) {
        Foo foo1 = new Foo("bar");
        Foo foo2 = new Foo("bar");
        Foo foo3 = foo1;
        System.out.println("foo1.equals(foo2): " + foo1.equals(foo2));
        System.out.println("foo1 == foo2: " + (foo1 == foo2));
        System.out.println("foo1 == foo3: " + (foo1 == foo3));
    }
}
```