Exam 1 Review

Lecture 5

Format

- The exam will be in two parts (across two days)…

1. Questions via Blackboard
   - ~20 questions
   - T/F, multiple choice/answer, output, legal code?

2. Coding via GitLab+Eclipse (think LA, sans resources)
   - JUnit tests
   - Individual methods, classes, **`main`**

- For each part you are allowed a single 8.5x11" piece of paper with whatever notes you wish
  - Can be different/same for parts 1 vs. 2
  - You cannot use other resources (e.g. prior programs, Google)
  - Bring your laptop charged!

Content

Everything we've covered all semester, including…

- Anything COMP1000
  - Command-line arguments
- Classes and Objects
  - UML
  - Static vs Instance, Visibility, **`final`** (x3)
  - Packages
  - Wrappers, **`ArrayList`**, **`StringBuilder`**/**`StringBuffer`**
  - String interning
- Inheritance and Polymorphism
  - Terms (+ abstraction, encapsulation)
  - **`this`**, **`super`**
  - Constructor chaining, dynamic binding
  - Overriding vs. overloading, annotations
  - Concrete vs. **`abstract`** classes/methods
  - Explicit vs. implicit casting
  - **`instanceof`**, **`equals`**, **`toString`**

Review Exercises

- The following slides contain exercises that will help you prepare for the exam
- The exercises give you an idea of the style of questions to expect as well as the complexity

Exercise

What is wrong with the following code?

```
public class Review {
   private int x = 10;

   public static void m() {
      x++;
   }
}
```

Answer

- The `m()` method is static (i.e. 1 per class), while **x** is an instance variable (i.e. 1 per object)

- Static methods only have access to static variables
  - Which object's **x** would we be accessing?

Exercise

What about the following code?

```
public class Review {
   private static int x = 10;

   public void m() {
      x++;
   }
}
```

Answer

- This code is just fine!

- All instance methods (i.e. 1 per object) have access to static variables (also known as “class variables”)

Exercise

- Write a **`Book`** class
  - Each instance has a title and author (constant once constructed), as well as getter methods, and overridden **`toString`**/**`equals`**
  - The class has a variable for the number of copies of any book sold (i.e. objects instantiated), as well as a getter

- Write a **`main`** that creates three books (a/b/a), outputs + compares, and outputs the number of books sold

Answer

```
final private String title;
final private String author;
private static int sold = 0;

public Book(String title, String author) {
      this.title = title;
      this.author = author;
      sold++;
}

public String getTitle() {
      return title;
}

public String getAuthor() {
      return author;
}

public static int getNumBooks() {
      return sold;
}

@Override
public String toString() {
      return String.format("'%s' by %s",
                           title, author);
}
```

```
@Override
public boolean equals(Object o) {
      if (o instanceof Book) {
            final Book b = (Book) o;
            return (b.title.equals(title) &&
                  b.author.equals(author));
      } else {
            return false;
      }
}

public static void main(String[] args) {
      final Book b1 = new Book("t1", "a1");
      final Book b2 = new Book("t2", "a1");
      final Book b3 = new Book("t1", "a1");

      System.out.printf("Books sold: %d%n",
                        Book.getNumBooks());
      System.out.printf("%s vs %s = %b%n",
                        b1, b2, b1.equals(b2));
      System.out.printf("%s vs %s = %b%n",
                        b1, b3, b1.equals(b3));
      System.out.printf("%s vs %s = %b%n",
                        b2, b3, b2.equals(b3));
}
```

Exercise

Inheritance vs. Polymorphism?

```
1. Object o = "hi";
2. public class B extends A {}
3. boolean b = x instanceof Foo;
4. (new Bar()).toString();
5. @Override
6. super
```

Answer

Inheritance vs. Polymorphism?

1. `Object o = "hi";`
   - Polymorphism: a subtype is being stored in a supertype for later use
2. `public class B extends A {}`
   - Inheritance: B inherits A methods/variables
3. `boolean b = x instanceof Foo;`
   - Polymorphism: the type hierarchy is being inspected at runtime, commonly before a type cast
4. `(new Bar()).toString();`
   - Polymorphism: dynamic binding chooses the appropriate method implementation to execute
5. `@Override`
   - Inheritance: annotation commonly used before providing a more specific method implementation
6. `super`
   - Inheritance: allows a subclass to access superclass methods/variables

Exercise

What is the output of the following code?

```
System.out.println(Integer.parseInt("101"));
System.out.println(Integer.parseInt("101",2));
System.out.println(Integer.parseInt("101",10));
System.out.println(Integer.parseInt("101",16));
```

Answer

```
101
5
101
257
```

Exercise

- True/False

  A class cannot contain both static and non-static methods

Answer

- False (e.g.see the **Book** class)

Exercise

What is the result of the following code?

A.java
```
package p10;

public class A {
     int i;

     public void m(int i) {
          this.i = i;
     }
}
```

B.java
```
package p10;

public class B extends A {
     public void m(String s) {
     }
}
```

1. *Compile error*
2. *Runtime error*
3. *Output?*

Review.java
```
package p10;

public class Review {
     public static void main(String[] args) {
          final B b = new B();
          b.m(5);
          System.out.printf("i is %d%n", b.i);
     }
}
```

Answer

```
i is 5
```

Exercise

On which line(s) do you find errors?

```
1. final int a;
2. int b;
3. Integer c = 200;
4. final String d = new String("hi?");

5. a = b = 100;
6. b = null;
7. c++;
8. c = null;
9. d.replace('?', '!');
```

Answer

On which line(s) do you find errors?

```
1. final int a;
2. int b;
3. Integer c = 200;
4. final String d = new String("hi?");

5. a = b = 100;
6. b = null; // only objects can be null
7. c++;
8. c = null;
9. d.replace('?', '!');
```

Exercise

- Write a method that returns the smallest integer from an **`ArrayList`** supplied as a parameter (or **`null`** if empty)

- Write a program that needs two command-line arguments: a *seed* and a *number*
  - Initialize a random number generator with the *seed* (first argument)
  - Create an **`ArrayList`** with *num* numbers from 1-100
  - Output the full list, as well as the smallest value (using your method above)

- Example runs…

```
$java p10.Review 100 10
[16, 51, 75, 89, 92, 67, 37, 89, 24, 14]
Smallest: 14

$ java p10.Review 101 10
[41, 91, 94, 55, 31, 66, 80, 34, 83, 7]
Smallest: 7

$ java p10.Review 100
Usage: <seed> <num>

$ java p10.Review a b
Usage: <seed> <num>
```

Answer

```
final private static String ERR_USAGE = "Usage: <seed> <num>";

public static Integer smallest(ArrayList<Integer> a) {
	if (a.isEmpty()) return null;

	int ret = a.get(0);
	for (int v : a) {
		if (v < ret) {
			ret = v;
		}
	}

	return ret;
}

public static void main(String[] args) {
	try {
		final int seed = Integer.parseInt(args[0]);
		final int num = Integer.parseInt(args[1]);

		final Random r = new Random(seed);
		final ArrayList<Integer> a = new ArrayList<>(num);

		for (int i=0; i<num; i++) {
			a.add(r.nextInt(100)+1);
		}

		System.out.printf("%s%nSmallest: %d%n", a, smallest(a));
	} catch (Exception e) {
		System.out.printf("%s%n", ERR_USAGE);
	}
}
```