WEEK FOURTEEN

Acknowledgements: Slides created based off material provided by Dr. Travis Doom

MISCELLANEOUS TOPICS

- None of the following are required, and you will not be tested on them:
  - Enumerated types
  - null
  - Encapsulation note
  - Copy constructors

ENUMERATED TYPES (ENUM)

- Pair a number (value) with a word (identifier)
- Very useful for encoding
- Makes code easier to read (good style)
- Each identifier in an enumerated type is an object of the type declared after the enum keyword
- Each identifier is ordered from 0 upwards

```
enum Color {RED, ORANGE, YELLOW,
GREEN, BLUE, INDIGO, VIOLET};

// RED is 0

// ORANGE is 1

// YELLOW is 2 …

// VIOLET IS 6

Color favColor = Color.BLUE;

if (favColor == Color.VIOLET)

      …
```

ENUM CONTINUED

```
enum Color {RED, ORANGE, YELLOW, GREEN,
BLUE, INDIGO, VIOLET};

Color favColor = Color.BLUE;
```

Output:

```
System.out.println(favColor);                        BLUE

System.out.println(favColor.ordinal());              4

System.out.println(Color.INDIGO.ordinal());          5
```

NULL

- Default value for any object reference variable before it is initialized
  - `Scanner scnr; // will be null until assigned a value`
- Keyword
- Can be stored in a reference variable
  - `String name = null;`
- Means that the variable currently refers to no existing object
- Good practice when writing methods to ensure that object reference parameters are not null

ENCAPSULATION NOTE

- Remember, we want to control access to our class fields
- If we write a getter for an object (not primitive type), what do we return?
    - If we return the reference to the actual field object, it can be modified even if it is private
    - Thus, we should return a *copy* of the object
    - This ensures that all the information is provided without the ability to change the class field
    - Essentially, we need to create a *new* object

COPY CONSTRUCTORS

- A constructor that has an object of the same class as a parameter
- Makes an identical copy or clone of the object

```
public class Course {
        private String name = “”;
        private int creditHours = 0;

        public Course(Course originalCourse) {
                this.setName(originalCourse.getName());
                this.setCreditHours(originalCourse.getCreditHours());
        }
}
```

SHALLOW COPY

```
public class Student {
        private ArrayList<Course> classes = new ArrayList<>();

        public Student(Student originalStudent) {
                for (Course c : originalStudent.getClasses()) {
                        classes.add(c);
                } // SHALLOW COPY: a reference to the Course is added, not a new separate object
        }         // If we modify the Course objects of the originalStudent, our new Student’s
 }                // Course objects would also change
```

DEEP COPY

```
public class Student {
        private ArrayList<Course> classes = new ArrayList<>();

        public Student(Student originalStudent) {
                for (Course c : originalStudent.getClasses()) {
                        classes.add(new Course(c));
                } // DEEP COPY: a new object is created and added to classes
        }         // If we modify the Course objects of the originalStudent, our new Student’s
 }                // Course objects would NOT change
```