SYSC 2004
Winter 2018
Lab 6

Objectives

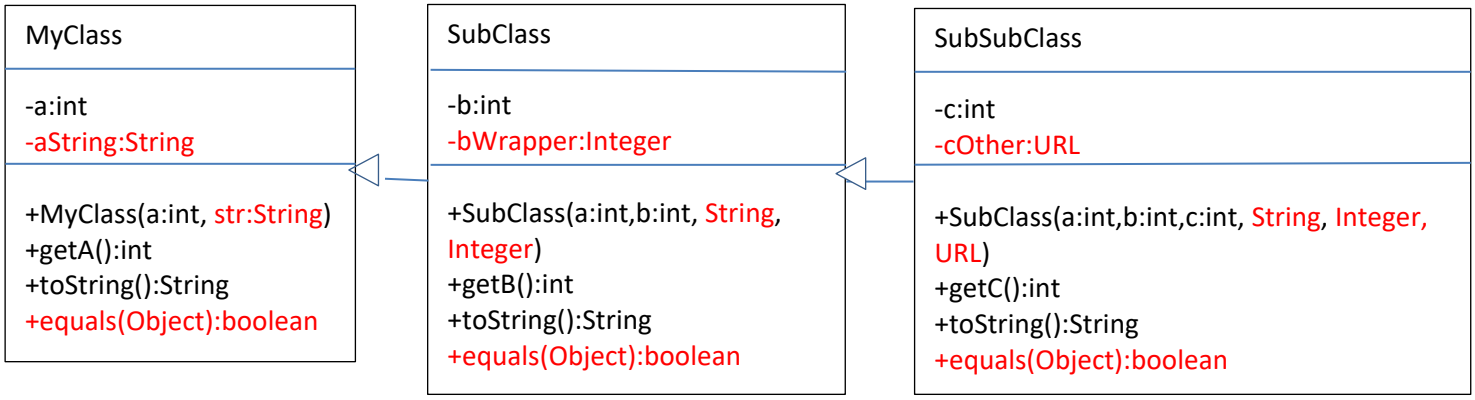
1. Practice with `equals()` along an inheritance chain
2. Practice with `HashMaps` (and incidentally, `Random`)

Submission Requirements: (Exact names are required by the submit program)

- **Without a submission you will not get a mark**
  MyClass.java, SubClass.java, SubSubClass.java, Lab6b.java, Patient.java, Lab6c .java

Part 1 –Inheritance and Class `Object`

To practice `equals()` and `toString()` along an inheritance chain, we will re-use the classes from Lab 4, with a couple of changes to bring in some additional complexity.



1. Within NetBeans, make a copy of your project Lab4. Call it **Lab6**. Delete the main class Lab4. You won't need it.
2. Re-write your constructors in all three classes to accommodate the new instance variables (shown in red), **following the rules for constructor chaining**.
   - Ask yourself: **Why** were these instance variables added? What is their importance to this exercise?
   - Don't be thrown off by the use of the class `java.net.URL`. It's just another class that could be very interesting to you. For now, just use one of the constructors available. Do a web search "**Java API URL**" and look at the list of constructors.
     - If you really, really need help, look at the solution at the bottom of this lab. But, **please** at least try it first.
3. Re-write your `toString()` methods for all three classes, **following the rules for inheritance chaining for `toString()` found in the lecture notes on class Object.**

- Format the string such that there is a / between variables from different levels of the inheritance hierarchy and a space between variables at the same level of the inheritance hierarchy For example, the `subsubObject` should print as following: "a = 8 SubSubClass / b = 9 77 / c = 10 http://www.google.com"

4. Write the `equals()` method for all three classes, **following the rules for inheritance chaining for `equals()` found in the lecture notes on class Object**
    - Work efficiently: After writing the `equals`() method for `MyClass`, copy-paste the code into `SubClass` and make just the necessary changes.
    - **Alternatively**, repetition is a form of study: Do **NOT** copy-paste and write each `equals`() method from scratch. By the end of the lab, you will have memorized the code and you're ready for the midterm!
5. Download the provided JUnit test code called **SubSubClassTest.java** and drag it into your Lab6 project. Run the tests until all tests succeed.

Part 2 – First Use of `java.util.HashMap`

It is not very object-oriented, but you will write a simple class containing only a static `main`() method in which you will do a series of `HashMap` operations, simply to demonstrate the use of a `HashMap`.

The HashMap will be used to store a `gradebook`. The `gradebook` has two columns: (1) the unique student `number` of a student and (2) the `grade` of that student. The `grade` is a traditional letter grade, ranging from A+ down to F.

- On an exam, you would be required to understand that this problem description translates to a variable called `gradebook` whose type is a `HashMap`, whose keys are `Integer` objects and whose values are `String` objects. Notice the use of the `New Courier font` to provide clues to you in this translation.

The `gradebook` is to be automatically filled with 1000 students. The 1000 students must each have a unique-but-random `number` between 0 and 10,000, and must have a random `grade`.

- On an exam, you would be required to understand that this problem description translates into a `for-loop` in which 1000 entries are put into the `gradebook`, and that both the `numbers` and `grades` are randomly generated using the `Random` class. You will be expected to be able to write this code.

Steps to follow (If you wish to practice for an exam, skip these instructions and do-it-yourself)

1. Declare the `gradebook` variable as follows:

```
HashMap<Integer,String> gradebook;
```

2. The variable is declared, but now you must instantiate it. Use `HashMap`'s default constructor.
3. Declare and initialized a random number generator, using `java.util.Random`.

```
Random random = new Random();
```

4. There is a little data-encoding problem to solve before we go further: The random number generator provides random integer values. Somehow, you have to convert those integer values to letter grades, of A+ or B- or C or F. Grades are `strings`, not characters (because we need A as well as A+). We will perform the conversion from integer-to-string using an array.

```
String grades[] =

{"F","D-","D","D+","C-","C","C+","B-","B","B+","A-","A","A+"};
```

   Example: A random integer of 0 corresponds to grades[0] which is an F.

5. Before the next step of actual coding, you also need to learn something about the `Random` class. It's time for you to do some reading:
   - Do a web search: "Java API Random"
   - Find the method `nextInt():int`
   - Find its **overloaded** version `nextInt (bound):int`
   - We want to use this overloaded version of `nextInt(bound)`. One time, we will use a bound of 10,000 to generate student numbers between 0 and 10,000. Another time, we will use a bound of 12 to generate a numeric grade of 0 and 12, which you will in turn use as an index into the `grades[]` array from the previous step (to convert 0 to "F" and 12 to "A+")
6. It is time to randomly generate 1000 entries into our `gradebook`.
   - Write the usual outline for a for-loop {} that runs 1000 times (for 1000 students)
   - **As a first draft:**
     - Randomly generate an integer value between 0 and 10,000 (called `number`) which will be the randomly generated student number.
     - Randomly generate an integer value between 0 and 12 (called `gpa`)
     - Use `gpa` as an index into the array `grades`, to convert to a `String` grade (called `grade`). This will be the randomly generated grade.
     - Put the `<number,grade>` pair into the `gradebook`.

- Print out the number of entries in the `gradebook`, using `HashMap::size()` method.
- Run your program. How many students are in your `gradebook`? Your for-loop was for 1000 students. Yet, you probably have less than 1000 students! **Why**?
    - Because the random number generator probably generated the same number twice, and your call to `HashMap::put()` over-wrote the first grade for that student with a second grade!
- **As the final version:** Look at your code. How can you re-write the code such that you generate 1000 entries? Such that if a duplicate number is generated, you discard it and generate another unique one instead? Such that if the gradebook already `contains` the generated number as a *key*, you don't use that number but instead you generate another different one? This is a simple exercise in loop logic combined with the `HashMap's containsKey()` method that you will be expected to know how to do on an exam.

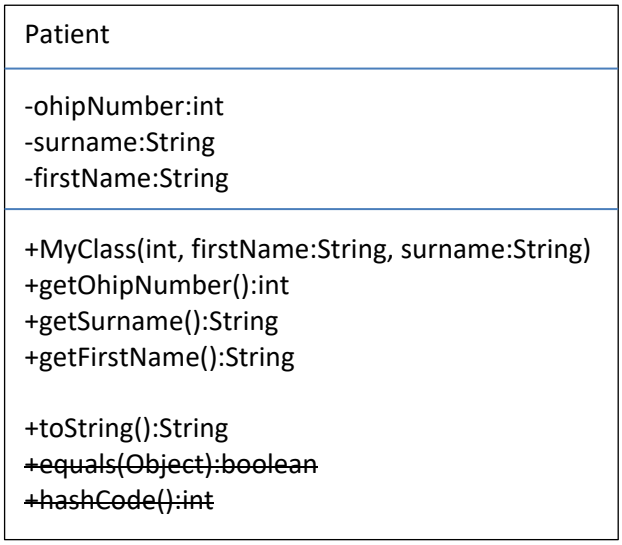
7. You may now wish to see all the entries in your `gradebook`. We won't have covered the code to do this efficiently in our lectures, so I'm providing it below. Please copy-paste into your program and let it run. If you have not used the exact variables names suggested, you will have to fix up the code. The syntax is called the Java for-each loop. Look it up in the textbook or online.

```
Set<Integer> keys = gradebook.keySet();
for (Integer stNum: keys) {
    String grade = (String) gradebook.get(stNum);
    System.out.println(stNum + " = " + grade);
}
```

Part 3 – Harsh Lessons in HashMaps

HashMap require that the classes used as keys and values have properly overridden two methods from class `Object`: **both** `equals`() and `hashCode`(). Let's see this in action.

We are going to build a hospital patient directory where we associate a patient with the hospital room. The patient will be encoded by the `Patient` class, below. The hospital room will be a `String`.



Steps:

1. Write the code for the `Patient` class such that it matches the given UML. Do **not** implement the `equals`() or `hashCode`() method **for now**.
2. Create a new class in your project called **Lab6c**. It will be the client class.
    a. Create a `HashMap` object called `hospital`.
    b. Create a `Patient` object call `p`, with a name of John Doe and an OHIP number of 1.
    c. Put `Patient p` in room **C123** (add this <key,value> pair to `hospital`)
    d. Now, ask `hospital` to search which `room` that Patient `p` is in. Print out the room value returned.
    e. Create a second Patient object called `other`, with the identical values to `p`.
    f. Use this second Patient object `other` to re-ask `hospital` to search for a room value. Print it out.
3. Run your program and pay attention to the results. Does the program work? (Probably not!, but why!)
4. Now, implement the `equals`() method (according to proper principles).
5. Without changing your client **Lab6c**, re-run your program with the new version of `Patient`, and pay attention to the results. Does the program work (Are the room numbers sensible? Probably not, but why?)
6. Now, implement the `hashCode`() method, returning the sum of the hashCodes of all its instance variables.

7. Without changing your client **Lab6c,** re-run your program with the new version of `Patient`, and pay attention to the results. Does the program work? (Hopefully yes, finally!)

Lesson: For a class to be used as a key in a hashMap, the class must implement both the `equals`() and `hashCode`() methods.
Lesson: To implement the `hashCode`() method, a class may delegate to the `hashCode`() method of its instance variables (i.e. in turn call the hashCode() method of the instance variables)

Solution For Part 1: You had to look! URL aURL = new URL("http://www.google.com");