OOP, static, generics, Associations

Instructors: Sam McCauley and Dan Barowy
February 14, 2022
OOP Continued
Testing the Student Class

• You should never write more than 10–20 lines without testing

• 4–5 is better

• Let’s test out our Student class
  • See some examples of making objects
  • How classes interact
public class TestStudent {

    public static void main(String[] args) {
        Student a = new Student(18, "Sam", 'B');
        Student b = new Student(19, "Bill L", 'A');
        // Some code to nicely print student details
        System.out.println(a.getName() + "", " + a.getAge() + ", " + a.getGrade());
        System.out.println(b.getName() + "", " + b.getAge() + ", " + b.getGrade());
    }
}
We can create as many Student objects as we need including arrays of Students.

```java
Student[] section = new Student[3];
section[0] = new Student(18, "Huey", 'A');
section[1] = new Student(20, "Dewey", 'B');
```
Final Student Array Example

```java
Student[] studentArray = new Student[4];
studentArray[0] = new Student(18, "Bill", 'B');
studentArray[1] = new Student(19, "Sam", 'C');
studentArray[2] = new Student(24, "Cathy", 'A');
studentArray[3] = new Student(20, "Dev", 'A');

//sort students
sortStudentsByGrade(studentArray);

//print students
for(int i = 0; i < studentArray.length; i++)
    System.out.println(studentArray[i].getName() + " : " +
                        studentArray[i].getGrade());
```
Objects and Special Methods
Classes are Types

- Remember: a class is really is to tell Java what kind of object we’re dealing with
- We’ll see later that one type may imply another
- Every Square is a Rectangle
- Every Student is a Person
- For now: every single object is also an Object
- What does that mean?
The Object class

- Object is a built-in class type in Java

- No instance variables!

- Three methods:
  - public String toString()
  - public bool equals(Object other)
  - public bool int hashCode() (we won't talk about this one until later)

- Every object is an Object, so every object has these methods!
toString()

- Returns a String representation of the object
  - (Sound familiar to the pythoners out there?)

- Cool part: if we System.out.println() an object, this gets called automatically

- Can we simplify our Student and TestStudent code with this in mind?
equals(Object other)

• How do we tell if two objects are equal?

• It’s going to depend on the object. For Student, we probably (only) check if their names are equal

• the .equals(Object other) method takes another object as input, and determines if the two objects are equal

• What happens if we use == to compare objects instead?
  • We would instead be comparing if the objects have the same memory address
  • I.e.: asks if it was created with the same new call
  • Let’s look at an example with Student

• Always use .equals(), not ==, when you are comparing objects!
The `Object` class

- Every object is also an `Object`
- How can we use this?
- One thing we can do: store any object as an `Object`
- If we have a stored `Object`, how can we interact with it?
- Only with `.toString()` and `.equals(Object other)`
- Let’s store some `Student` objects as `Object` and see what happens
- Notation for casting: put the type in parentheses
- I.e.: `Object newObj = (Object) s1;` stores `s1` as an `Object`
Writing a `.equals` method for Student

Let’s check if two students are equal

- Challenge: the argument to `.equals()` is an `Object`

- We want to check if the `name` is the same, but `Object` type does not have a `.getName()`

- Solution: transform the other object into a `Student` first!
public class Student {
    // instance variables
    private int age;
    private String name;
    private char grade;

    // A constructor
    public Student(int age, String name, char grade) {
        // What would age, name, grade
        // refer to here...?
    }
}

• Answer: it refers to the most local version
public class Student {
    // instance variables
    private int age;
    private String name;
    private char grade;

    public Student(int age, String name, char grade) {
        this.age = age;
        this.name = name;
        this.grade = grade;
    }
}

- this keyword specifies the current object (like self in python)
- Lots of strong feelings about the above syntax. You can use it if you want
- Some people *always* use this to refer to instance variables in Java. You don’t have to unless you think it’s clearer.
static variables and methods
**static variables**

- A *static* variable is a property of the *entire class*, not a single object.
- In other words: there’s only one *copy* of the variable.
- We saw: each *Student* object has its own *name* variable.
- Let’s add a static variable *occupation* to the class. (There is only one *occupation* for all *Student* objects).
- What happens when we change *occupation*?
- To access a static variable, use the name of the *class* directly: `Student.occupation`
static variables continued

- Can access `static` variables without creating an object of the type at all!

- Can set their values when they are declared.

- Can I also use the constructor to set their values?
  - ...yes. But you *probably* don’t want to.
  - After all, you may use a static variable before the constructor is called
static methods

- Like static variables: property of the entire class rather than a specific object
- Can be called without creating an object of the class!
- Java rule: you cannot access non-static variables from a static method. Why??
  - The non-static variables are created when an object is created
  - The static method may be called before any object is created!
  - So if the static method accessed them, they may not exist yet! (Big problem)
- Why is main static?
Let’s look at some examples

- First, CoinStrip

- Let’s make a Triangle.java class to store a triangle