More Classes

Today's topics:

- `final` keyword
- Scope public/private
- Classes / Objects / Constructors (and constructor overloading)
- toString
- Static and instance

This sounds like a lot, but they are all tied together!

final countdown, err frontier.. variable!

- the keyword `final` when placed before a variable declaration
  - prevents the variable from being overwritten
  - which means for primative variables
    - that value is set, and unchanging
  - for objects
    - it means it will always point towards that object in memory
    - but the inner contents can change
- Think of final as *readonly*

Why final?

It is meant to be a **compile time** check. Your code won't compile if you try to overwrite the final variable!

How does it look?

In [2]:
```
public class SillyClass {
    public static final String NAME = "Silly String";
    public static String NO_NAME = "No name";
    public final String localName = "Silly putty";

}

// the following would be inside a main method usually
System.out.println(SillyClass.NAME);
SillyClass egg = new SillyClass();
System.out.println(egg.localName);

SillyClass.NO_NAME = "Yes name";
System.out.println(SillyClass.NO_NAME);
```

```
Silly String
Silly putty
Yes name
```

But the following would not compile!

In [3]:
```
SillyClass.NAME = "hello";
egg.localName = "Slinky";
```

```
|   SillyClass.NAME = "hello";
cannot assign a value to final variable NAME
```

Wait, that is a public variable?

For the most part, we have:

- Made methods public
- variables private

But, we just made a variable **public**

Both variables and method can be

- public
- private
- protected (future class)
- internal/package private (future class)

Public

public means all classes have access. This controls the way other classes interact with the code you are writing.

Traditionally, we want to control access!

However, if variable is final - they can't mess with it, so safe to make it public (if we want to allow access).

Private

Most instance variables are private.

Many **methods** can be private.

- This means only the 'class' can see the private method.
- This is often called a helper method (helps other methods work). Used a lot in good design.

We actually won't use it much, as we can't grade private methods!

In [ ]:
```
public class SillyString {
```

```
    public int sprayLength() {
        if(checkSprayRemaining()) {
            return 6;
        }
        return 0;
    }

    private boolean checkSprayRemaining() {
        return true;
    }
}

SillyString blueCan = new SillyString();

System.out.println(String.format("Shoots a spray of %d feet!", blueCan.sprayLength()));
```

```
Shoots a spray of 6 feet!
```

In [ ]:

```
System.out.println("Does it have more? " + blueCan.checkSprayRemaining());
```

```
|   System.out.println("Does it have more? " + blueCan.checkSprayRemaining());
checkSprayRemaining() has private access in SillyString
```

Building Objects: Constructors

In the past, when we have added properties to objects, we have called methods.

```
public class Rectangle {
    private int width;
    private int length;

    public void setWidth(int width) {
        this.width = width; // remember this!
    }

    public void setLength(int length) {
        this.length = length;
    }

    public static void main(String[] args) {
        Rectangle small = new Rectangle();
        small.setWidth(10);
        small.setLength(12);
    }
}
```

But does a rectangle make sense with **zero** length and width?

- not really.
- We ideally want a way to set the length and width at creation time.

Enter Constructors!

Specialized methods are invoked only when **new** is used.

Properties:

- they don't have a return type defined
- they can only be called using **new**
- the job is to set values that are *required* for the class to work
- you can have more than one with different parameters
    - it will call the one with the matching parameters

In [ ]:
```
public class Rectangle {
    private int width;
    private int length;

    public Rectangle(int width, int length) {
        setWidth(width);  // i could have typed this.width = width
        setLength(length); // better practice is to call setters if they exist
    }

    public void setWidth(int width) { this.width = width; }

    public void setLength(int length) {this.length = length;}

    public int getWidth() { return width;}
    public int getLength() { return length;}
}
```

In [ ]:
```
Rectangle smallBuilding = new Rectangle(10, 12);
String formatted = String.format("Rectangle - width: %d, length: %d",
    smallBuilding.getWidth(), smallBuilding.getLength());

System.out.println(formatted);
```

```
Rectangle - width: 10, length: 12
```

Also, once a constructor is made, the "default" constructor can't be used until you purposely add it!

In [ ]:
```
Rectangle unknownBuilding = new Rectangle();
```

```
|   Rectangle unknownBuilding = new Rectangle();
constructor Rectangle in class Rectangle cannot be applied to given types;
  required: int,int
  found:    no arguments
  reason: actual and formal argument lists differ in length
```

In Class Activity

In Canvas you will find a link to the code. I have placed the code **both** in zybooks and on the github.

- Feel free to use whatever is easier / quicker for your table!

You will see StudentInClass does not have a constructor, and will not compile as is.

Add a constructor, and run the program.

The format of the constructor should be

- classID
- ename
- name

> Discussion
> Why are some methods private?
> Why was I able to keep classID public?
> What pick a 'path' through the program, and try to follow the method calls based on client input.

Example Path Trace:

Run program

- Creates an InclassMain object
- Invokes the run() method
  - Asks for client input [Enter: "l"]
  - Reads response using scanner (whole line!)
  - invokes/calls processAction("l")
    - in **processAction**
    - compares "l" with "list" || (OR!) "l" - finds it to be true
      - invokes the printStudents() method
      - in **printStudents()**
        - invokes printStudent with the object amy
          - in **printStudent(amy)**
          - String.format

            ```
            Class: CS164
            Name: Amy Pond
            Grade: 0.00%
            ```

          - prints String to the screen (return)
        - invokes the printStudent with the object rory
          - see above, different output
        - invokes the printStudent with the object clara
          - see above, different output
        - invokes the printStudent with the objet alice
          - see above, different output
        - return out of printStudents()
      - return out of processAction()
    - repeats loop, asks client for next action
    - type exit
  - ends loop
- end program

This can vary for everyone! This is just one way to think about it

toString()

Whenever we have wanted to print code, we have built a 'formattedOutput' type method.

What happens if we just print the object?

In [ ]:
```
public class Student {
    private final int id; // we want it internal to student only
    public final String ename; // ok for other classes to see, but not change it!
    private String name;

    public Student(int id, String ename, String name) {
        this.id= id;
        this.ename = ename;
        setName(name);
    }

    public void setName(String name) {this.name = name;}
    public String getName() {return name;}
}

Student airbender = new Student(811222333, "zackeis", "Aang");
Student waterbender = new Student(899888777, "maewhit", "Katara");
Student weaponmaster = new Student(877666555, "jackdese", "Sokka");

System.out.println(airbender);
System.out.println(waterbender);
System.out.println(weaponmaster);
```

```
REPL.$JShell$12$Student@7fca74fa
REPL.$JShell$12$Student@69467ee0
REPL.$JShell$12$Student@7c5252b6
```

That doesn't make sense!

But if you think about classes, it does.

- with every **new** key word, I create an additional memory table.
- so for the program, the memory tables look like the following.

Current "Stack"

variable	value
airbender	Student@7fca74fa
waterbender	Student@69467ee0
weaponmaster	Student@7c5252b6

Student@7fca74fa

variable	value

variable	value
id (final)	811222333
ename (final)	zackeis
name	Aang

Student@69467ee0

variable	value
id (final)	899888777
ename (final)	maewhit
name	Katara

Student@7c5252b6

variable	value
id (final)	877666555
ename (final)	jackdese
name	Sokka

Essentially, Java is printing what is on the stack memory!

- Only because you didn't tell it differently
- Introducing `toString()` a way to tell it do print something else for Objects

In [ ]:
```java
public class Student {
    private final int id; // we want it internal to student only
    public final String ename; // ok for other classes to see, but not change it!
    private String name;

    public Student(int id, String ename, String name) {
        this.id= id;
        this.ename = ename;
        setName(name);
    }

    public void setName(String name) {this.name = name;}
    public String getName() {return name;}

    public String toString() {
        return String.format("Character: %s, ID: %d", getName(), id);
    }
}

Student airbender = new Student(811222333, "zackeis", "Aang");
Student waterbender = new Student(899888777, "maewhit", "Katara");
Student weaponmaster = new Student(877666555, "jackdese", "Sokka");

System.out.println(airbender);
System.out.println(waterbender);
System.out.println(weaponmaster);
```

```
Character: Aang, ID: 811222333
Character: Katara, ID: 899888777
Character: Sokka, ID: 877666555
```

- With toString() we can print objects directly
  - and they make sense / are readable to us!!

In Class Activity: Task 2

Add a toString() method to `StudentInClass.java`.

The method should return a String of the following format:

```
Class: CS164
Name: Amy Pond
Grade: 89.00%
```

where class is the class they are in, name is the name of the student, and grade is the *current* grade.

You can pull the formatting String from InclassMain.printStudent

You will also want to update the InclassMain.printStudents() method to be the following:

```
System.out.println(amy);
System.out.println(clara);
System.out.println(rory);
System.out.println(alice);
```

Run the program, if done correctly - nothing should look different!

Memory (Static / Instance)

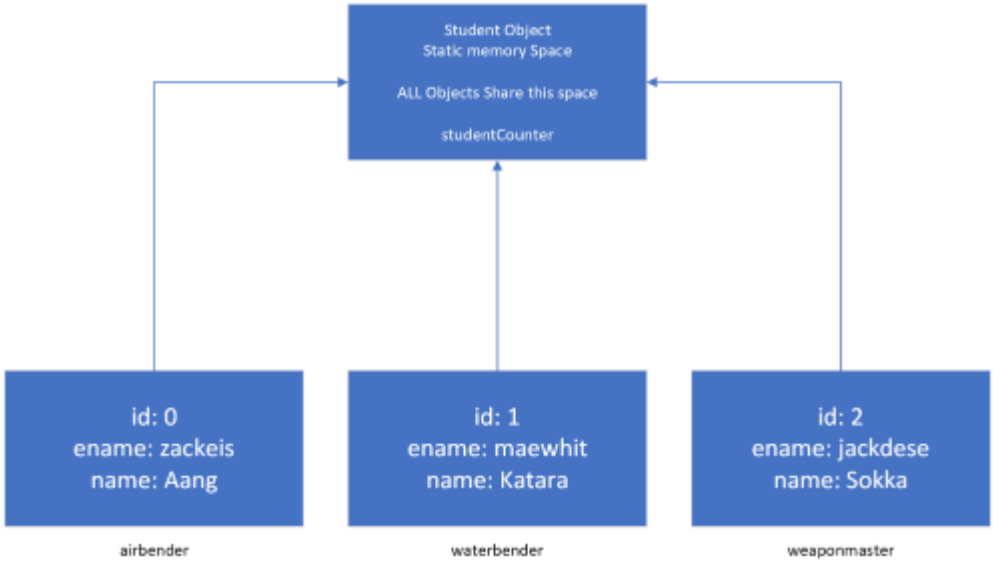
Let's think about memory.

In the example above:

- `new` created a new instance of the object
- we built out the memory table
- the variable values are unique to that object

Static:

- Anything that is static is not unique
- It is *shared* across all objects
- A way to visualize this is the following:



In [ ]:
```java
public class Student {
    private final int id; // we want it internal to student only
    public final String ename; // ok for other classes to see, but not change it!
    private String name;
    static int studentCounter = 0;

    public Student(String ename, String name) {
        this.id= studentCounter; // just set the ID based on the student object create
        this.ename = ename;
        setName(name);
        studentCounter += 1;
    }

    public void setName(String name) {this.name = name;}
    public String getName() {return name;}

    public String toString() {
        return String.format("Character: %s, ID: %d", getName(), id);
    }
}

Student airbender = new Student("zackeis", "Aang");
Student waterbender = new Student("maewhit", "Katara");
Student weaponmaster = new Student("jackdese", "Sokka");

System.out.println(airbender);
System.out.println(waterbender);
System.out.println(weaponmaster);

System.out.println(Student.studentCounter)
```

```
Character: Aang, ID: 0
Character: Katara, ID: 1
Character: Sokka, ID: 2
3
```

Notice:

ID continues to change. If we continue to run the code, it will continue to go up! (at least until the program ends!).

Conclusion

Just focus on building objects, and "drawing out" what is going on.

The design of when to use static and instance makes more sense with practice.