Review 2

- Code, code, code
- Code, where does it go?
- Main method?
- Coding with ***style***
- Homework

Code, code, code

Let's review some fundamentals that will help you every time you write a program.

Where does code go?

In order to speak English properly, we need to understand English grammar. Grammar helps us know how to arrange our words into sentences so others can understand us correctly.

Java has grammar rules too! We need to follow the rules of Java's grammar so that *the computer* can understand us correctly.

Let's talk about the most important parts of the grammar, from smallest to biggest.

Statements

A statement in Java is like a *single sentence*. Here are some examples of statements:

```
score = hwScore;    // Setting a variable to a value.

greet();            // Calling a methods
System.out.println("My score is " + hwScore);

return score;       // Returning a value in a method.
```

Notice that each line looks a little bit different. Each contains logic, and logic is what your program is made of.

These are rules that when learned, you can use to know what any line of code is doing.

Variable declaration and initializatio.

Variable declarations can only go in methods.

```
    String firstName;
//  ^      ^         ^
//  Type   Name      Semicolon

    firstName =     "Ford Prefect";
//  ^         ^     ^              ^
//  Name      Equal Value          Semicolon

    short age = 30;
//  ^     ^     ^
//  Type  Name  Value

    Movie starWars = new Movie("A New Hope");
//  ^     ^           ^
//  Type  Name        Value
```

Field declaration and initialization

Field declarations can only go in classes. Follow the rules for *Variable declarations and initialization* with some exceptions:

```
   protected String movieName;
// ^         ^      ^
// Access    Type   Name
// Modifier

   public   static short numberOfMovies = 0;
// ^        ^      ^     ^                 ^
// Access   Static Type  Name              Value
// Modifier
```

Method declarations

```
   public    String  createGreeting(String friendName) {
// ^         ^       ^                ^
// Access    Return  Name             Arguments
// Modifier  Type

     String greeting = "Hey " + friendName + "!";
//   ^
//   Body

     return greeting;
//   ^
//   Return statement
   }
```

Note: Unless your method has a return type of `void`, your method is required to have a `return` statement.

Conditions and Loops: if statement

```
    if (age >= 16) {
// ^    ^
// If  Condition

      issueDriversLicense();
//    ^
//    Body
    } else if (age == 15) {
//    ^          ^
//    Else If  Condition

      learnersPermitOnly();
//    ^
//    Body
    }
```

The sections in between the curly braces `{}` are run when the condition in the parentheses `()` is true. The code inside the curly braces *will also be statements* (the body).

Conditions and Loops: while loop

```
   while (age < 16) {
// ^       ^
// If      Condition

     noDriversLicenseForYou();
//   ^
//   Body
   }
```

Conditions and Loops: for loop

```
   for (int i = 0; i < 100; i++) {
// ^    ^          ^         ^
// For  Initialization      Step
//                 Condition

     displaySystemStatus();
//   ^
//   Body
   }
```

Classes

```
   public    class   Hangman {
// ^         ^       ^
// Access    Class   Name
// Modifier

     protected String word;
//   ^
//   Field declarations

     public boolean isDone() { /* ... */ }
//   ^
//   Method declarations
   }
```

A Full Example

```
public class Student {
  private String firstName;
  private String lastName;
  private int score;

  public String greet() {
    return "Hi, my name is " + firstName;
  }

  public int setHomeworkScore(int hwScore) {
    greet();
    score = hwScore;
    System.out.println("My score is " + hwScore);
    return score;
  }
}
```

Parts of the previous example

- This file contains the `Student` class
- The `Student` class contains some field declarations and method declarations
- There are three fields: `firstName`, `lastName`, and `score`
- There are two methods: `greet` and `setHomeworkScore`

What are our rules?

- A class declaration can contain:
  - Field declarations
  - Method declarations
- A method declaration can contain:
  - Statements
- A statement can be one of:
  - Variable declaration/initialization
  - Method call
  - `if`, `if` / `else` statements
  - `for` / `while` loops
  - `return` statement

The Main Method

A special method declaration!

```
public static void main(String[] args) {
  System.out.println("Hello world!");
}
```

What are the parts?

```
    public   static void    main(String[] args) {
// ^        ^      ^        ^    ^
// Access   Static Return   Name Arguments
// Modifier        type

      System.out.println("Hello world!");
//    ^
//    Body
    }
```

Why do we need a main method?

The main method is the entry point into your program. It is like the front door of your house. You can't get into your house without a door!

Your program can have lots of reusable parts, like *classes* and *methods*. But it needs to start from a single specific place. By having a `main` method, you tell Java where to start.

Where does the main method go?

The `main` method is just like any other method declaration, so it goes inside of a class declaration.

Example: Pet Store

- `Main.java`
- `PetStore.java`
- `Animal.java`
- `Food.java`

`Main.java`

You could put the main method in the `Main` class, like this:

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Welcome to the pet store!");

    PetStore myStore = new PetStore();
    myStore.openForBusiness();
  }
}
```

Running your code - what happens?

Then you call it from the terminal like this:

```
> java Main
```

Whatever class you pass to the `java` command, it must have a `main` method since this is where Java will start from.

So this means, *"Hey Java, start running my program, and the starting point is in the `Main` class!"*. Or, *"Hey Java, go into my house, and here is where the front door is!"*

How many `main` methods do I need?

One. If you have multiple `main` methods, Java will only run one of them for you, and it will be the one that you tell it to.

```
> java Main
> java PetStore
> java YoutubeServer
```

The three commands above will run three different `main` methods found in three separate locations: `Main.java`, `PetStore.java`, `YoutubeServer.java`. And it will only run one of them per command!

Why just ***one*** main method?

Your house only needs one front door! You *could* have a main method defined in multiple classes, but you still have to tell Java *which one to use* when you run your program.

Quiz

If I have a `Main.java` file with the following contents and I run the command at the bottom of the slide, what will I see print to the screen?

```java
public class Program {
  public static void main(String[] args) {
    System.out.println("Running Program.main");
  }
}

public class Main {
  public static void main(String[] args) {
    System.out.println("Running Main.main");
  }
}
```

```
> javac Main.java
> java Main
```

Coding with ***style***

- What is style in code?
- Is it easy for you to read?
- Is it easy for others to read?

As professional programmers, most of your time will be spent reading a line of code rather than writing it. Or at least you will have to read it more times than it took to write it in the first place, so make sure it is easy to read.

Remember, you will read your code more often than you will write it.

How can we make our code easy to read?

There's a few techniques programmers follow to make their code more "readable" and preferences can differ from programmer to programmer, but there are a few common methods that are expected of everyone.

Always use brackets

The first rule we'll place is that you should always use curly brackets (`{` / `}`) even when they are optional.

```
if (isBankAccountOwner())
  System.out.println("Accessing bank account information
  showPrivateBankTransactions();
```

```
if (isBankAccountOwner()) {
  System.out.println("Accessing bank account information
  showPrivateBankTransactions();
}
```

Indentation

Second, proper indentation is another easy rule to follow to have good style.

Which of the two following examples is easier to read? What is the code doing?

```
public class Main {
public static void main(String[] args) {
for (int i = 0; i < 100; i++) {
if (i % 5 == 0 && i % 3 == 0) {
System.out.println("FizzBuzz");
} else if (i % 3 == 0) {
System.out.println("Fizz");
} else if (i % 5 == 0) {
System.out.println("Buzz");
} else {
System.out.println(i);
}
}
}
}
```

```java
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 100; i++) {
      if (i % 5 == 0 && i % 3 == 0) {
        System.out.println("FizzBuzz");
      } else if (i % 3 == 0) {
        System.out.println("Fizz");
      } else if (i % 5 == 0) {
        System.out.println("Buzz");
      } else {
        System.out.println(i);
      }
    }
  }
}
```

Whitespace

What is whitespace? Whitespace are new lines, tab characters, and space characters. Adding whitespace in your code can help you create visual "blocks" and "sections."

Adding whitespace in a line of code

```
int a = 7;
int b = 20;
int c = 2*(a-3)+4*b-2*(a-b-3)+5;
```

```
int a = 7;
int b = 20;
int c = 2 * (a - 3) + 4 * b - 2 * (a - b - 3) + 5;
```

Adding whitespace between statements

```
if (age >= 60) {
  activateSeniorDiscount();
}
if (gender.equals("Female")) {
  provideFemaleShoes();
} else if (gender.equals("Male")) {
  provideMaleShoes();
}
```

```
if (age >= 60) {
  activateSeniorDiscount();
}

if (gender.equals("Female")) {
  provideFemaleShoes();
} else if (gender.equals("Male")) {
  provideMaleShoes();
}
```

Why?

Why is this important? As far as your computer is concerned, it's not. Indenting your Java code won't alter it's behaviour, neither will adding spaces in between the plus operator, so why? Because you will read your code more often and more times than write it, and so we follow these style guides to make it easier for humans (including your future self) to read.

Homework

You're learning a lot and it's hard to remember everything, we understand. But we also understand that it takes a lot of practice to learn and remember everything you need in order to become an effective programmer, so this week we're going to assing more homework as a way to give you more chance to practice what you're learning.

What do we expect?

- We have example text output for each assignment; your program's output should look *exactly* like ours.
- Follow the code style guides you learned today.
  - put brackets where needed
  - indent your code
  - add whitespace to add visual sections to your code
- Every assignment should be in a *separate file* that is *ready to compile and run* with no changes.

Additional resources

- Introduction: https://youtu.be/TBWX97e1E9g
- User input: https://youtu.be/yYN8u90MKCg
- Conditionals (if/else/switch): https://youtu.be/qZ2pb6BIjLk
- Loops (for/while): https://youtu.be/efvZmFd1prA
- Methods: https://youtu.be/1HTsLK_m2ao
- Classes: https://youtu.be/rGIJiUO-dZA
- Logic behind programming and building a game: https://youtu.be/_pUz-GJcdRU, https://youtu.be/bQTFXIZWzKw
- Arrays and collections (ArrayList): https://youtu.be/eNPX2pTiaHI, https://youtu.be/lEqvmsqjpT0