Week03 - Primitive Data Types and Objects

In class: Gaddis 2B slides.

Primitive Data Types

- `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` and `char`.
- Great explainer of Java Data Types from w3schools.
- We will focus mostly on `int`, `double`, `char`, and `Boolean`.

Non-Primitive Data Types

Non-Primitive Data Types are called Java Objects or Java Classes.

- String, Scanner, etc. are non-primitive data types (objects).
  - The Java API gives us these objects (for free!).
- In Java, you make your own data types using **classes**.
- Java is known as an Object Oriented Programming (OOP) language.
  - Objects are a big deal in Java!

Back to Primitives

Float/double

- Both are decimal-based primitive data types.
  - Mostly you should use **double**.
  - Float uses less memory (half) than double so it can be (slightly) faster.
  - Using Double, you can represent scientific notation with e or E: `double sci = 4.728197e4;` equals `47281.97`

Char

- enclosed with Single Quote.
- For example, 'A' for A.
- Supports Unicode (extended characters, every language) with "\u".

    - So If you type ' \u4F60', you get this char 你.
    - However Google's Java style guide says, just type the character. You don't need to type the \u code.
- Slide 9: Char also supports Hexadecimal. For example, 0x6e (no quotes).
    - 0x07 is kind of fun: it says BEL.
    - 0x20 can be used for explicit spacing.

Variable Assignment and Initialization

- You can assign variables without initializing with data up top, but you can also just doing it in one line.
- Tip on Assignment 2a:
    - For the seven days and total, initialize variables without assigning data.
    - For the three more (halfDay, etc.), initialize them WITH their value. How much should a halfDay be worth? How much should a fullDay be worth?
- Remember: variables in Java **should** always have a **type**! (`int` , `double` , `String` , `Scanner` , etc.)
- HOWEVER, the `var` keyword was introduced to Java in 2018 in Java version 10. This means you rely on the Java compiler to understand what type of data you are storing in a variable. This can result in compiler errors if YOU (the programmer) don't have a grasp on which types you are working with.
    - Examples: `var str = "Hi"; var num = 100; var doub = 123.45`
    - I don't encourage this! because Java is (or used to be) a **Strongly Typed Language**
    - Other languages have `var` ! It used a lot in **weakly typed languages** such as JavaScript and PHP (In Python, you don't even have to say `var x = 20` , you just say `x = 20`) .
        - However, some people decided JavaScript needed to be fixed by adding types and created TypeScript , which is now one of the best ways to write JavaScript.

Arithmetic operators

- Binary operators (+, -, etc.) - BI because they require two operands. With these operators, you combine two numbers.
- Operand is the value on which the operator acts:
    - 2 and 2 are operands in 2 + 2.
- Be very careful with Integer division!
  It rounds down, so you'll end up with non-precise answers:

`jshell> 6 / 3`
`2`
`jshell> 7/3` 2`
- Order of operations: 6/2*(1+2)
  - Generally follows PEMDAS.
  - JShell helps to test equations!
- Combined Assignment Operators
  - `+=` and `-=` are used frequently in Loops.

Casting - Convert a Type to Another

- Be sure to read and understand Section 2.7: Conversion between primitive data types.
- Casting can be done implicitly from Smaller to Larger (Widening) data types:
  `byte → short → char → int → long → float → double`

  So you can convert an int to a double with no problems:
  `int num = 100;` doublt result = num result = 100.0`
- Explicit Casting (Narrowing Conversing) must be done when converting a larger data type to a smaller because there is a risk of losing information or precision.
  You must manually specify the target type in parentheses before the value.
  `double decimal = 100.99`
  `int decCast = (int) decimal;`
  `decCast = 100`

JShell

- JShell is super easy and useful Java console within Terminal (mac), PowerShell (Windows), or Command Prompt (Windows).
  - Comes with Java!
  - Run calculations, declare and use variables, etc.
  - Helpful video about JShell.

Constants with `Final`

- `final` keyword creates an unmodifiable variable.
  - Used for constants that will never change. Example: `PI`.
  - You should write in all caps with underscores between words: `DAILY_PARKING_RATE`.

- Tip: If you use `final` on assignment 2a for the parking rates correctly, I'll give you a couple extra points.

More about Non-Primitive Data Types

String

- `String` is a non-primitive data type. Java gives us this object, it's not native to the machine code of the computer. Briefly:
  - We create Variables. (`int x = 25; print x;`)
  - The computer stores those somewhere in memory (example, memory address `0x001`)
  - For Primitive data types, the value itself is stored in that memory location.
  - For non-primitive data type, that memory location is a reference to other locations which store the primitive data types.
    - Let's say `String S = "Sarah"` is stored in computer memory at `0x002`. If you looked inside that memory space, you'd see a reference to another place in memory (`0x116`) along with a length.
    - A String is made out (primitive type) Chars. So if I look in `0x116` location, I see `S`. Then `0x117`, I see `A`. Etc..
    - When I tell Java, `print S`, it looks in `0x002` drawer, gets the length of Chars and prints the chars in order.
- I said String is non-primitive and that Java **gives** us this.
  - Java has objects that are written in Java that you can use. It's a major benefit over coding directly on the machine in Assembly or C.
  - Thousands of objects given to us by the Java API.
    - Examples: Scanner, Array, Web Request, Server, Image Handler.
  - You write your own using classes.
  - There are many benefits to non-primitive data types. Like Methods!
- But first, Strings are special non-prim types because they can be assigned as a String literal.
  - `String value = "Hello"`
- All other java non-prim types must be created using the `new` keyword.
  - `String value = new String("Hello");`

Methods

- Perform an action on Java objects.
- Specified by dot notation after an **instance of the class**, followed by parentheses with parameters (if required) or empty parentheses: `object.method()`
  - "Instance of a class" is a difficult concept! See below.
- Example: String objects have a .length() and .replace(*searchChar*, *newChar*) methods.

  `String s = "Sarah"` String newString = s.length()`
  (length doesn't require any parameters)

  `String b = s.replace('a', 'b')`
  (replace takes two Char data types as parameters. The first one 'a' is the character to search for in `s`, the second one is the new character 'b' to replace the search char if it is found)

  if you print `String b`:
  `b = Sbrbh`
  `
- Scanner objects have a nextLine() method
  `Scanner keyboard = new Scanner(System.in)`
  `String input = keyboard.nextLine()`

Scanner

- Our first non-primitive data type that behaves like other non-primitive data types!
- We know `System.out` is used to print to console, Scanner is `System.in` is used to type to console.
- Must be imported at top of file: `import java.util.Scanner;`
- Must be initialized using the `new` keyword:
  `Scanner keyboard = new Scanner (System.in);`
- Accepts user input using `keyboard.nextLine()`, `keyboard.nextInt()`, `keyboard.nextDouble()` .. depending on the type of expected input.
  - **Important!** If the Scanner is expecting an Int, and you type a letter, your program will crash.

"Instance of a Class"

Java Objects must be initialized like variables must be initialized.

With primitives, initializing a variable is easy. You just tell Java you want a variable, and it assigns a space for it on your computer's memory:
`double dbl;`

You can also initialize the variable while putting data in the value, and Java stores that data on your computer:
`double dbl = 123.45;`

Since String is non-primitive, the variable stores a reference to another memory location.
`String s = "Sarah";`
If you ask your computer to show you `s`, you would see a pointer to another location where you can find the actual primitives `S`, `A`, `R`, `A`, `H`. They would be in order, and the `s` variable also know how many characters are stored in those other locations.

Another way of initializing that String variable is like this:
`String s = new String("Sarah");`
^ This is the way that every other object is initialized.

String is a special object that has two ways to be initialized

- (it's the most common non-primitive data type. And it's relatively straightforward: it is a just sequence of Chars (primitives) of a certain length).

The usual syntax to initialize an object is like this:
`Class variablename = new Class()`
Classes can have parameters, which are written inside the parentheses, like this:
`Class variablename = new Class(ClassParameter1, ClassParameter2)`

Here's how we initialize a Scanner object:
`Scanner keyboard = new Scanner (System.in);`

If I write my own class called `Dog`, and it has a parameter for dog breed (a string), here's how I would initialize an Dog object:
`Dog myPugDog = new Dog("pug");`

In the above examples: `s`, `keyboard`, and `myPugDog` are INSTANCES of their classes. They have been initialized and we can use methods to work with them.

We can't say `Scanner.newLine()`. Methods don't work on **the class itself**. We use the method on **the instance of the class** that has been initialized with the `new` keyword (unless it's a String). So, in our example, we will say `keyboard.newLine()` to use the `newLine()` method on our Scanner object.

`Parse` methods

Remember how we said that Java objects have methods? We also said that Primitive Data Types are not Java Objects. Therefore, Primitive Data Types must not have methods right?

Java thought about that! (Thank you, James Gosling)

Java has its own wrapper class for the primitive types to make them objects. From the Class Integer documentation:

> The `Integer` class wraps a value of the primitive type `int` in an object.

These Java Wrapper Classes for primitives are called using the class names `Integer`, `Double`, `Character`, and `Boolean` (there are wrappers for the other primitives too).

- Take note that these wrapper classes start with a capital letter (like all objects, e.g., String), whereas primitive keywords are lower-cased.
- Also note that the wrapper classes use the whole word while the primitives are abbreviations (e.g., `int` vs. `Integer`, `char` vs. `Character`).

These wrapper classes allow Java to use methods on primitive data types. One example are the Parse methods. `parseInt` and `parseDouble` are useful for extracting numbers out of strings!

You access them by referencing the wrapper class, and calling the method on a local variable (in this example, we want to extract the numbers (ints and doubles) out of the String `str`):

```
int number;
String str;
double dbl;

str = “100.25”;
number = Integer.parseInt(str);
dbl = Double.parseDouble(str);
```

You can also convert a integer or double into a String using the `String.valueOf()` method.

```
int num = 100;
String str = String.valueOf(num); // "100"
```

Math Class

Chapter 2b had a quick slide about the `Math` class. So I want to mention it.

Like the wrapper classes, you don't need to make an instance of the Math class.

- (i.e., you don't need to say `Math myMath = new Math();`)

This is because Math and the wrapper classes have **Static Methods** available in addition to **Instance Methods** (chapter 4 or 5 material).

So similar to the wrapper classes, you can use methods available to Math. In this example, you can get the square root of 64 and the value of 2 to the power of 8:

```
double square = Math.sqrt(64);
double exponent = Math.pow(2,8)
```

Here's a list of All Math Methods from w3schools and a bit more information about Java Math from the same source.