KEY

Week 1	Exercise 1	Whose Type is it Anyway	20 Minutes

Reminders:

- Write your names in large print on the paper. Instructors want to meet you and address you by name. Help us out by writing your name so we can read it over your shoulder and talk to you personally.
- Think out loud! Use the paper. Show us your ideas.
- Share the pen! Both partners need to interact with each other and the paper. If it isn't written down, you may not receive credit for participation. Your paper needs to show two distinct handwritings for full credit.

Goals:
Meta-types, classification of Java types with respect to meta types, introduction to containers

Description:
There are two **meta-types** in Java: **primitives** and **non-primitives**.

Primitive types are the predefined and basic types such as `int`, `char`, `float`, `boolean`, etc. A **primitive type** holds only one value and no more.

A **non-primitive type** holds collections of data and has the capacity to contain multiple, independent values. In other words, a non-primitive type requires a container to hold multiple, associated values. The term **reference type** will be closely associated with non-primitive types, but it has a slightly different meaning.

Tasks:
In the following code examples, classify each variable as either primitive and non-primitive type depending on whether it can be represented as a single supported value or it requires a collection to represent it.

1.

```
width = 7.5
height = 5
rect = rectangle(width, height)
perimeter = (width + height) * 2
area = width * height
```

2.

```
msg = “hello world”
for i = 0 : len(msg)
    ch = msg[i]
    ord = ch - ‘a’
    print(ord)
```

3.

```
data = sizeof(int) * 10
i = 0
x = 0
while i < 10
  data[i] = i
  x = x + data[i]
```

Questions:

1. How can you identify the difference between a primitive and non-primitive in these examples?
   Variables assigned a value are primitives while variables assigned to structures and non-primitives. For example, `width = 7.5` *appears to be assigning a float to the variable width and in Java we would expect width to be declared as* `float width`*. Alternatively,* `rect = rectangle(width, height)` *appears to be assigning something that is more complex than a single value to the variable rect and in Java rect would be declared as* `Rectangle rect` *which is clearly a custom class type. All custom classes are non-primitives.*
2. What if this was syntactically correct Java code rather than pseudocode? Are there some clear indicators in the Java language that you are dealing with a primitive or non-primitive type?
   If the variable is declared with one of the 8 Java primitive data types it is a primitive. If it is declared with anything other than a primitive type, it is non-primitive and it is represented using a reference type. The significance of this difference will become more clear later, but for now, you need to be able to recognize primitive and non-primitive.
3. Is the Java `float` type a primitive type or a non-primitive type?
   There are 8 Java primitive types: `byte, short, int, long, float, double, boolean, char`*. If a variable is declared of one of these types, it is a primitive type. If it is declared of any other type, it is non-primitive. Other languages may support more or less primitive types, consult the relevant documentation on that language.*
4. Is a Java String a primitive type or non-primitive type?
   A String is non-primitive. The give away in Java is that by convention custom types, i.e. classes, are capitalized.
5. Why are Java class names capitalized? It is a clear convention among the Java developer community, so what is the reasoning behind capitalizing the classname.
   Recall that Java, and many other languages, is case sensitive meaning that the symbol `String` *is different than the symbol* `string`*. By capitalizing class names, it leaves available the same letters for use as a variable if the variable is lowercase. This allows a programmer to easily relate a variable to its type, using the same letters, but having different symbols, the capital letter indicates a type and the lowercase indicates a variable.*
6. Similarly, why are Java variable names not-capitalized? Again, it is a clear convention among the Java developer community, so what is the reasoning?

As with the previous question, when reading code, a developer can easily identify variables because they are lowercase. It helps the developer easily identify the context of symbols in their code. If the symbol begins with a lowercase letter it must be a variable and if the symbol begins with a capital letter it must be a custom type.

7. What is the default value for all primitives?
   All primitives are initialized by Java to 0. This may seem nonsensical at first if you think that a boolean must be `true` or `false` and cannot be 0, but consider that the type determines how the value stored in it is interpreted. Everything in memory is stored as a number, but the type determines how that number is translated and interpreted. A `false` boolean in memory is stored as a zero. This means that type determines the context of the value stored so it is critical information. If zero is interpreted as `false`, then what is `true`?
8. How do we represent "nothing" and what can have that value?
   A primitive can never be "nothing". It must always have a value, so this does not apply to primitives at all. Try not to confuse the idea of nothing with a primitive as there is no representation of nothing for primitives. Non-primitives are something else entirely. When declaring a non-primitive, you are declaring a reference to an object. However, that does not mean you are creating the object as that is a separate step. So a reference type can refer to "nothing". We call this special value `null` in Java. A `null` reference means you have declared a non-primitive variable that points to nothing. If you try to access a `null` reference, i.e. nothing, Java will error because there is nothing to access.

Prompts (If a team is doing particularly well, you can ask these questions):

- What range of values can be stored in a selected primitive type? For example, can an `int` hold any number? If not, what are its limits? *No, an int can hold a lot of values, but not every integer value. The size of an int is fixed, so it has a specific range of possible values. For a 32-bit system, this range is [-2^31, 2^31-1]. It is about -2 billion to +2 billion so there are limits that need to be considered for problems involving extreme values.*
- Can two primitive types hold the same values? *Yes, floats can represent integral (int) values. However, floats are prone to errors in accuracy, so it using one type versus the other should be informed by the problem you are working on.*
- A String is a container right (per the definition), so how is the data really represented for a string? *A string is really a character array. The String type in java is designed to hide this as much as possible so that all String types appear to be a unique type. The String class is used to hide the array and collect all String operations into one definition.*
- How much space is needed to store the String type. *Since a String is a character array, the space depends on the number of characters in the string. Each character is a byte by definition, so the amount of space needed is at least* ***length of string * size of a byte***

Conclusions:

- A memory location can only store one thing at a time. It cannot store two things in one location. We would need two locations instead to store two things.
- For one of your container types, how many locations would you need? How does the computer keep track of all these locations for a custom type?
- Primitive types must have a value, so primitives can NEVER be null
- Container types may be declared before the space they contain is created. This means that a container type may refer to nothing, so non-primitives can have a null reference.

Further Information:

Java Primitives : https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
Java Naming Conventions :
https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
Pseudocode : https://en.wikipedia.org/wiki/Pseudocode