CS330: Programming Language Project (PLP)
Assignment 3: Data types and naming conventions

In Java, data types are classified into two categories: primitive and non-primitive data types. There are eight types of primitive data types: *byte, short, int, boolean, char, long, float* and *double*. Non-primitive data types typically include *strings, arrays,* and *classes*. The main difference between primitive and non-primitive data types are that primitive data types are defined in Java while the non-primitives are not defined but created by the programmer instead (Java Data Types, n.d.). Another difference are that non-primitive data types all have the same sizes, however for the size of the primitives, it depends on the data type itself.

Examples of some Java data types:

• Float

```
//float example
float num = 3.1415269;
System.out.println(num);
//prints 3.1415269
```

• Boolean

```
//boolean example
class booleanDataType{
public static void main(String args[]){
boolean Java = true;
boolean Python = false;
System.out.println(Java);
//prints true
System.out.println(Python);
//prints false
}
}
```

• Array

```
//array example
class MyArray{
public static void main(String args[]){
String[] grades = {"A", "B", "C", "D", "F"};
int[] myNum = {10,20,30,40};
System.out.println(grades[0]);
//prints A
System.out.println(myNum[1]);
//prints 20
}
}
```

• Int

```
//int example
int num = 2020;
System.out.println(num);
//prints 2020 true
```

• String

```
//string example
class helloWorld{
public static void main(String args[]){
String greeting = "Hello World";
System.out.println(greeting);
//prints "Hello World"
}
}
```

• HashMap

```
//hashMap example
class MyClass{
public static void main(String args[]){
HashMap<String, String > people = new
HashMap<String, String >();
people.put("John", "30");
people.put("Maria", "24");
people.put("Lu", "44");
System.out.println(people);
//prints {John = 30, Maria = 24, Lu = 44}
}
}
```

Discussion Questions:

1. What are the naming requirements for variables in your language? What about naming conventions? Are they enforced by the compiler/interpreter, or are they just standards in the community?

The naming conventions in Java are not forced by the Language itself from its compiler, however it is highly encouraged by several Java communities such as Sun Microsystems and Netscape. One of the standard naming convention for Java is the use of CamelCase (GeeksforGeeks, 2020). CamelCase is the practice of writing words without spaces or punctuation of any kind, an indication of the start of the next word is the use of a single capital letter, an example of CamelCase *LooksLikeThis*!

Java communities have also described the naming conventions of variables. The naming requirement for variables should typically be written in lowerCamelCase, which is the use of the CamelCase letter except the first letter is lowercased. Variable names should also be "short yet meaningful," where it is fashioned to indicate to a casual observer the intent of the variable (Sun Microsystems, 2015). One word variables should normally be avoided, however there are some widely practiced exceptions such as the use of the variable declared as i for integer, and a variable declared as c for characters (Sun Microsystems, 2015). From what I have seen, the choice of naming a variable to with an underscore _ or a dollar sign $ is slightly debatable because some communities like Sun Microsystems discourage it while GeeksforGeeks encourage it (Sun Microsystems, 2015; GeeksforGeeks, 2020).

The practice of following these naming conventions make Java programs much easier to analyze and read for a casual observer of the code. I have created examples of how variables are named in Java below.

```
int i = 0; // single i for the use of integer variable
char c = 'a'; //single c for the use of a character variable
int milesPerHour = 0; // example of lowerCamelCase
```

2. Is your language statically or dynamically typed?

Before I discuss how Java is typed, I will briefly go over what it means to be statically or dynamically typed to fully understand where Java lies. For a language to be statically type, it means that the language would perform type checking at compile time, as well as declare data types for variables prior to using them (Oracle, 2015). Dynamically typed languages would perform type checking at runtime, and there is no need to set the variables to a data type (Oracle, 2015).

In the case of Java itself, it is statically typed because it is required to declare the data type of the variable before its use. In the example below, there is a variable called num that is an int data type with the value 2020 and we can tell just by looking at the code.

```
int num = 2020;
System.out.println(num);
```

3. Strongly typed or weakly typed?

Strongly typed and weakly typed are two terms for programing languages, and there are certain characteristics that would make a language lean towards the description as strongly typed or weakly typed. A strongly typed language would typically have strict rules at compiler time, which would result in more errors occurring during compilation (Gries, 2018). While a weakly type language would have looser regulations such as creating a random variable $a = 5$. A strongly type language would also utilize a variable in respect to its type, an example is that you would not see a Boolean to preform arithmetic operations (Gries, 2018).

Java as a programming language is strongly typed because each variable must be assigned with a data type. The strong typing of Java "ensures that no value can be interpreted as something that it is not" (Gries, 2018). To go back on the brief example $a = 5$, the only way for this to be accepted in Java is that a is declared as an integer where, *int a = 5,* would work perfectly for Java. Since Java is considered as strongly type it would guarantee more safety compared to other weakly typed languages.

4. If you put this line (or something similar) in a program and try to print x, what does it do? If it doesn't compile, why? Is there something you can do to make it compile? x = "5" + 6

The only way to make Java x = "5" + 6 to make x a string data type with the value of *"'5' + 6,"* I have placed an example of the program I have typed below. What happened within this program is that the int 6 is concatenated to the string 5, which resulted in a new string 56. Any other way of describing x as a non-primitive data type such as an int or float would not worked because of Java is strongly typed.

```
class myClass{
public static void main(String args[]){
String x = "5"+6;
System.out.println(x);
//prints the new string 56
}
}
```

5. Describe the limitations (or lack thereof) of your programming language as they relate to the coding portion of the assignment (adding ints and floats, storing different types in lists, etc). Are there other restrictions or pitfalls that the documentation mentions that you need to be aware of?

There are a few limitations regarding data types that Java has built within the language, knowing these limitations would help us understand boundaries and restrictions that Java carries for these data types.

Byte Limitations:

One limitation of the byte data type is that it is bound to the values between -128 to 127, anything outside of the range would produce an error (Jarosciak, 2018).

```
class myClass{
public static void main(String args[]){
byte x = 128;
}
}
//error var 'x' is of an incompatible type
```

Char Limitations:

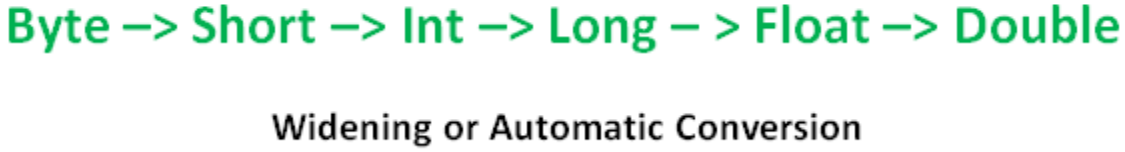
The char data type is only capable of 16-bit entities, so the number of characters it can take in is quite limited, which is an issue because over-time Unicode scalars has evolved and can support characters over 16-bits (Jarosciak, 2018).

6. How do type conversions work in your language? Are the conversions narrowing or widening, and do they work by default or do they have to be declared by the programmer?

When you need to assign one primitive data type to another Java has two types of methods for data conversion and this process is known as Type Casting. There are two types of primitive data types are compatible. Java will follow with Automatic Type Conversion, if not compatible the data types will need to be converted explicitly (Vaidya, 2019).

Automatic Conversion (Widening):

Automatic Conversions is known as widening because follows the conversion of smaller primitive data types into larger primitive data types. The process of Widening is automatically done by Java. The image below illustrates the process of widening (GeeksforGeeks, 2019).



Widening or Automatic Conversion

```
int i = 200;
long l = i; //automatic type conversion
long f = l; //automatic type conversion
```

Output:

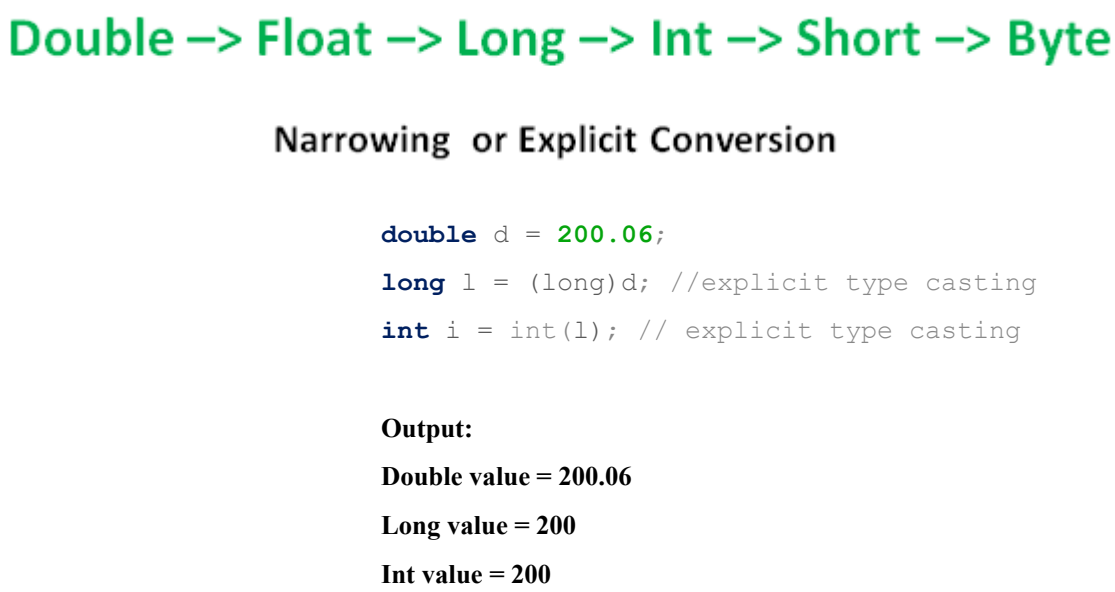
Int value = 200

Long value = 200

Float value = 200.0

Explicit Conversion (Narrowing):

Explicit Conversions, also known as narrowing, is quite the opposite of widening, since the process follows the conversion of larger primitive data types into smaller primitive data types. In the process of narrowing, it needs to be done manually by the programmer, where the programmer needs to assign a double into an int as an example. The image below illustrates the process of narrowing (GeeksforGeeks, 2019).



Works Cited

Gries, D. (2018). *Safety and Strong Versus Weak Typing*. Retrieved September 25, 2020, from https://www.cs.cornell.edu/courses/JavaAndDS/files/strongWeakType.pdf

Sun Microsystems. (2015). *Naming Conventions*. Retrieved September 25, 2020, from https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

GeeksforGeeks. (2019, November 17). Type conversion in Java with Examples. Retrieved September 25, 2020, from https://www.geeksforgeeks.org/type-conversion-java-examples/

GeeksforGeeks. (2020, August 24). *Java Naming Conventions*. Retrieved September 25, 2020, from https://www.geeksforgeeks.org/java-naming-conventions/

Jarosciak, J. (2018, January 18). Java for Beginners 2 – Addressing Data Type Limitations. Retrieved September 25, 2020, from https://www.joe0.com/2018/01/18/java-for-beginners-addressing-data-type-limitations/

Java Data Types. (n.d.). Retrieved September 24, 2020, from https://www.w3schools.com/java/java_data_types.asp

Oracle. (2015). Dynamic typing vs. static typing. Retrieved September 24, 2020, from https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html

Sun Microsystems. (2015). *Naming Conventions*. Retrieved September 25, 2020, from https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html

Vaidya, N. (2019, July 25). *What is Type Casting in Java?: Java Type Casting Examples*. Retrieved September 25, 2020, from https://www.edureka.co/blog/type-casting-in-java/