Allocating Memory to Variables

Consider the following fragment of java code

```java
int a;
int b;
```

When this program is compiled memory is allocated to both variables. Since `a` and `b` are of type int and this is a primitive type in Java, the amount of memory allocated to `a` is large enough to store any value given to this variable; similarly for the memory allocated to `b`. 
Allocating Memory to Variables

We can think of the memory of the computer as a group of cells where we can store values. Each cell has a unique address that can be used to access it. Each cell, for example, might consist of 1 byte (or 8 bits).
Allocating Memory to Variables

For example, if \( a \) is allocated to address 100 and \( b \) is allocated to address 150, the computer’s memory will look like this:

\[
\begin{array}{c}
100 \\
4 \text{ bytes}
\end{array}
\quad
\begin{array}{c}
150 \\
4 \text{ bytes}
\end{array}
\]

\( a \) and \( b \) are assigned each a block of 4 bytes because in Java an \textit{int} has a size of 4 bytes. The first byte allocated to \( a \) is in address 100, the second one in address 101, and so on. Java keeps track of where the variables are stored in memory in a table called the \textit{symbol table}. 
Allocating Memory to Variables

If now the following code is executed:

```java
a = 3;
b = 15;
```

The computer’s memory will look like this:
Allocating Memory to Variables

Non-primitive variables are handled in a different manner. Consider the following Java code:

```java
public class Person {
    private String firstName, lastName, email;

    public Person(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public String getFirstName() { return this.firstName; }
    public String getLastName() { return this.lastName; }
}
```
Allocating Memory to Variables

Consider the following Java code:

```java
Person p;
p = new Person ("John", "Doe", "jd@uwo.ca");
```

When the declaration of `p` is processed (statement `Person p;`), a block of memory is allocated to `p`, say starting at address 400 and large enough to store a reference to an object of class `Person`:

```
400
null
```
Allocating Memory to Variables

By default Java stores the value `null` in each non-primitive variable when it is declared. When the object is created:

```java
    p = new Person ("John", "Doe", " jd@uwo.ca");
```

a block of free memory large enough to store the above object of the class `Person` (large enough to store the `String` values for `firstName`, `lastName`, and `email` and the methods of the class `Person`) is allocated to this object and the values “John”, “Doe”, and “jd@uwo.ca” are stored in it. Let this block of memory start at address 500.

Note that the object is not stored in address 400, which was allocated to `p`. Instead in address 400 the computer stores the address 500 of the above object. The computer’s memory will now look like this:
Allocating Memory to Variables

Object of class Person

```
getFirstName()
getLastName()
```

```
"John"
"Doe"
"jd@uwo.ca"
```
Allocating Memory to Variables

Variable p is called a reference variable, as it does not store an object, but a reference or an address of an object. To access the content of the object referenced by p in Java we use the dereferencing operator ".".

So, for example p.firstName has the value "John" and p.lastName has the value "Doe". Alternatively, invoking the method p.getFirstName(); will return the string "John" and invoking the method p.getLastName(); will return the string "Doe".