In-class laboratory A: Draw the memory

Programming Fundamentals 2

9th March 2021

Goals

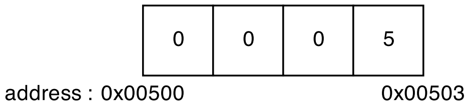
- ✭ Understand the memory representation of Java objects.

Memory in Java

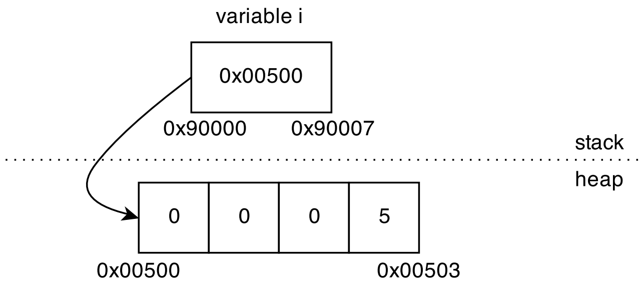
The memory is organized linearly, and you ask block of adjacent memory through the operator `new`. We use the following class as an example:

```
public class Integer {
  private int x;
  public Integer(int x) { this.x = x; }
}
```

What happens in memory when we execute `Integer i = new Integer(5);`? First, we reserve a memory zone of sufficient size to contain an integer, coded on 4 bytes:



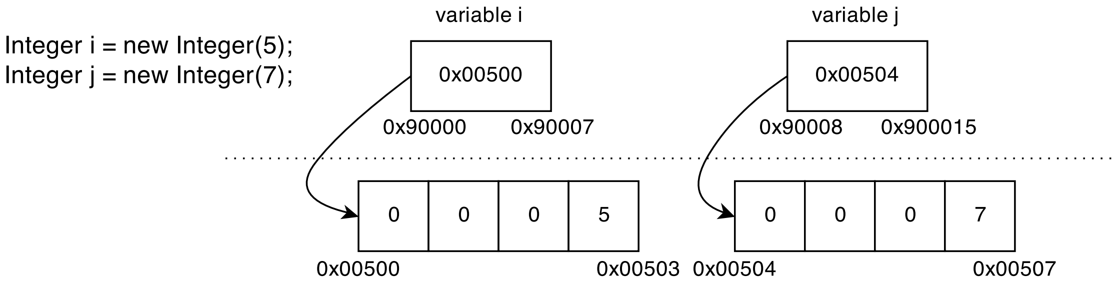
But that's not all, in Java, every variable containing an object use an *indirection*, which means that the variable contains the address of the allocated memory zone, so we have:

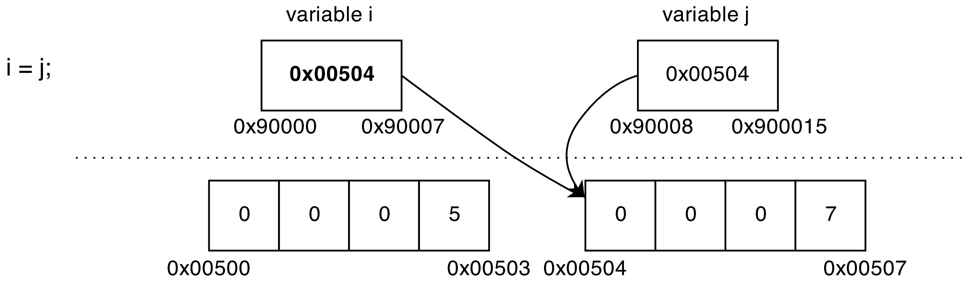


We note that memory is divided into two: the *stack* on one side, and the *heap* on the other side. Every variable is stored onto the stack, but the memory zone towards such variables points to can be allocated onto the heap. You must remember, that in a program, we only access to the heap through a variable containing a heap address, but that the address of this variable is always in the stack.
Suppose we have the following code:

```
Integer i = new Integer(5);
Integer j = new Integer(7);
i = j;
```

What happens in memory? As we can observe on the next diagram, the variable i refers to the same memory zone than j, which means we can modify a same object through two variables:





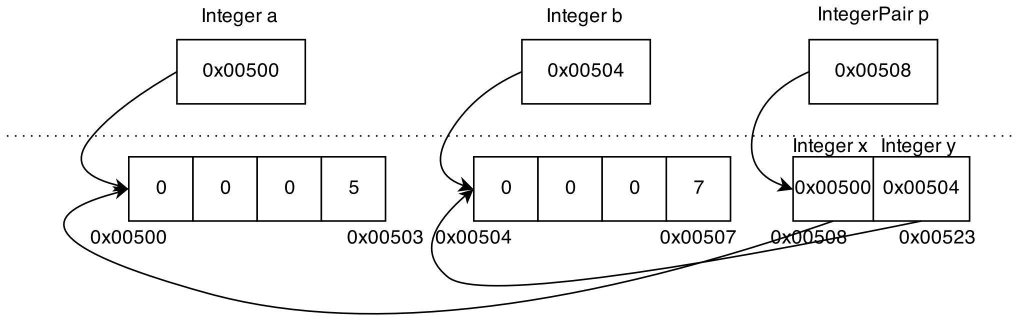
The memory zone pointed by i is now inaccessible, we can never use it again and the *garbage collector* will clean this zone and make it accessible again later.

Object's attributes can also point to other objects, consider the following code:

```
public class IntegerPair {
  private Integer x;
  private Integer y;
  public IntegerPair(Integer x, Integer y) {
    this.x = x;
    this.y = y;
  }
}

Integer a = new Integer(7);
Integer b = new Integer(5);
IntegerPair p = new IntegerPair(a, b);
```

We represent the memory of this object in the next diagram. Notice that the attributes x and y points to the same location than a and b.



Finally, we must distinguish between primitive types (`int`, `double`, `char`, `...`) and objects (`IntegerPair`, `String`, `ArrayList`, `...`) because primitive types do not request heap memory, but are automatically allocated on the stack. Consider `int i=9; int j=2; j=i;`, the value of i is copied in j, and not an address, as it would be the case for objects. Therefore, we will obtain two distinct elements i and j, and modifying one will not change the other. Note that for *copying object*, you must use the method `clone`, which must be manually implemented for the corresponding object.

Exercise 1 – Draw me the memory!

The answers to the exercises are diagrams of the memory, possibly with additional textual explanations.

1. Represent the memory for the variables i and j at the end of the following program:

```
int i = 9;
Integer j = new Integer(i);
```

2. Represent the memory for the variable $numbers$ at the end of the following program. Note that an array consists in adjacent memory cells.

```
int numbers[] = new int[6];
numbers[3] = 99;
```

3. Supposing that `String str = "abc";` is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Represent the memory for the variables $name$, $subname$ and $subname2$ at the end of the following program. You can and should consult the Java documentation for the methods on `String`.

```
String name = new String("Giselle");
String subname = name.substring(2, 4);
String subname2 = name.clone().substring(1, 3);
```

4. Represent the memory for the variables me and $mother$ at points (a) and (b).

```
public class Person {
  private Person mother;
  public Person() { mother = null; }
  public my_mother_is(Person p) { mother = p; }
}

Person me = new Person();
Person mother = new Person();
// (a)
me.my_mother_is(mother);
// (b)
```

5. Represent the memory for the variable $person$ at points (a), (b) and (c) and of the variable p at point (b).

```
public Person {
  private int age;
  public Person(int age) {
    this.age = age;
  }

  static void make_new(Person p) {
    p = new Person(9);
    // (b)
  }
}

Person person = new Person(1);
// (a)
Person.make_new(person);
// (c)
```