Is Java Pass-by-value?

<https://www.cs.virginia.edu/~jh2jf/courses/cs2110/java-pass-by-value.html>

Our book, and the internet, says that Java is pass-by-value always. What does that mean exactly, and why does Java appear to be pass-by-reference? Well, let’s start by looking at the definitions:

**Pass by value**: makes a copy in memory of the parameter’s value, or a copy of the contents of the parameter. Here is an example in Java:

public static void main(String[] args) {

...

int y = 5;

System.out.println(y); // prints "5"

myMethod(y);

System.out.println(y); // prints "5"

}

public static void myMethod(int x) {

...

x = 4; // myMethod has a copy of x, so it doesn't

// overwrite the value of variable y used

// in main() that called myMethod

}

**Pass by reference**: a copy of the address (or reference) to the parameter is stored rather than the value itself. In this case, modifying the value of the parameter will change the value. Here is an example in C++, which is *very* similar to Java syntax (cout is C++’s way of printing to the terminal):

int main() {

...

int y = 5;

cout << y; // prints "5"

myMethod(y);

cout << y; // prints "4"

}

// & below tells C++ to specifically "pass x by reference"

int myMethod(int &x) {

...

x = 4; // myMethod has the address of x, or the

// reference to x (rather than the value),

// so it overwrites the value of variable y

// used in main() that called myMethod

}

So, why does it appear that Java is (mostly) pass-by-reference, but books say that Java is always pass-by-value? It is in the nuance of conceptualizing what Java does and how best to understand that at this stage in our discussions.

Let’s look at another example, one that conceptualizes Java as pass by reference. Below is the “Cat” class that contains the name of our Cat. We will then look at an adoption method that changes the name of a given cat:

public class Cat {

private String name;

// Setter

public void setName(String name) {

this.name = name;

}

// Getter

public void getName() {

return this.name;

}

}

public class Example {

public static void adoptCat(Cat c, String newName) {

c.setName(newName);

}

public static void main(String[] args) {

Cat myCat = new Cat();

myCat.setName("Charlotte");

System.out.println(myCat.getName()); // prints "Charlotte"

adoptCat(myCat, "Lydia");

System.out.println(myCat.getName()); // prints "Lydia"

}

}

Passing in the Cat myCat into adoptCat() and changing the name there using the parameter c did, in fact, change the name of the cat! This intuitively looks like the pass-by-reference example above: making a change to the passed-in object affects that object’s values in memory. We can think of Java *intuitively as pass-by-reference for all objects*. This is why we professors called Java pass-by-reference.

Java is officially *always* pass-by-value. The question is, then, “what is passed by value?” As we have said in class, the actual “value” of any variable on the *stack* is the actual value for primitive types (int, float, double, etc) or the reference for reference types. That is, for a reference variable, the value on the stack is the address on the heap at which the real object resides. When any variable is passed to a method in Java, the value of the variable on the stack is copied into a new variable inside the new method.

For primitive types, this should make sense. Let us consider the original example from earlier with ints:

public static void main(String[] args) {

...

int y = 5;

System.out.println(y); // prints "5"

myMethod(y);

System.out.println(y); // prints "5"

}

public static void myMethod(int x) {

...

x = 4; // myMethod has a copy of x, so it doesn't

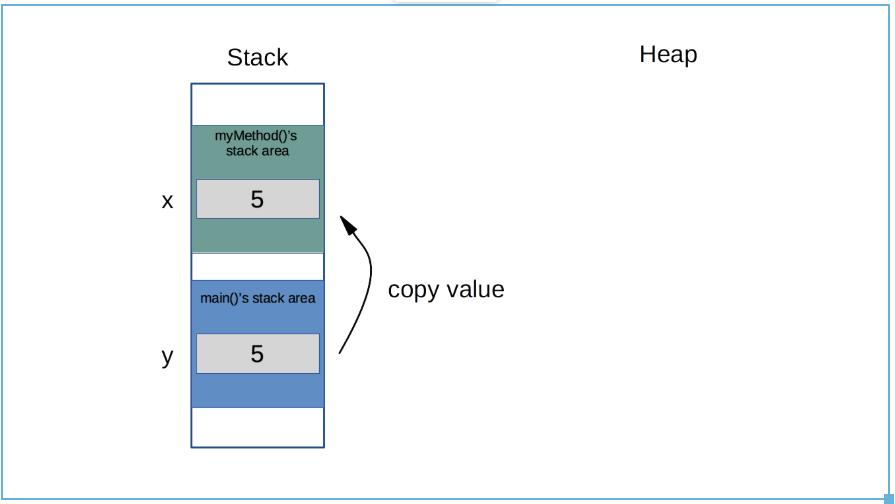
// overwrite the value of variable y used

// in main() that called myMethod

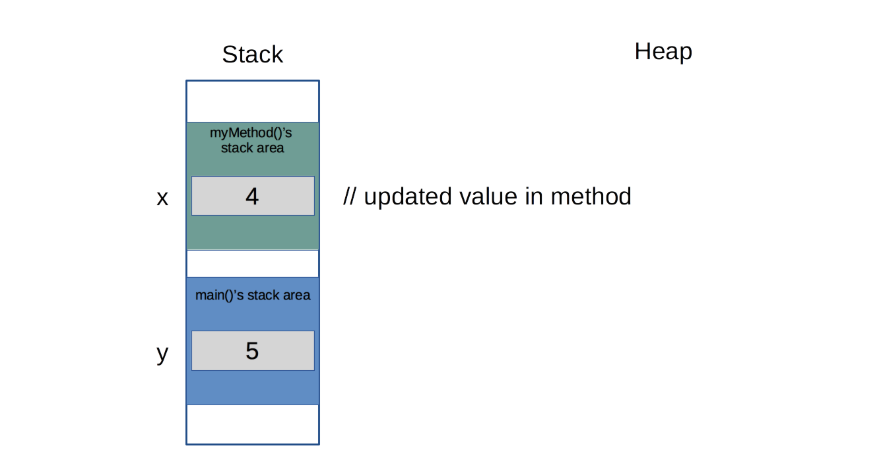
}

In memory, this will look like the following:

Call myMethod which creates a copy of y’s value inside variable x on the stack.

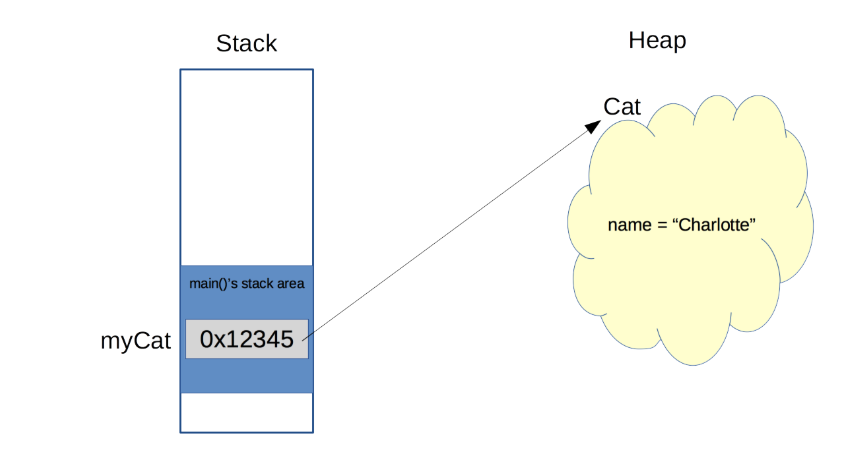


myMethod set’s x = 4, changing x’s value while leaving y’s untouched.

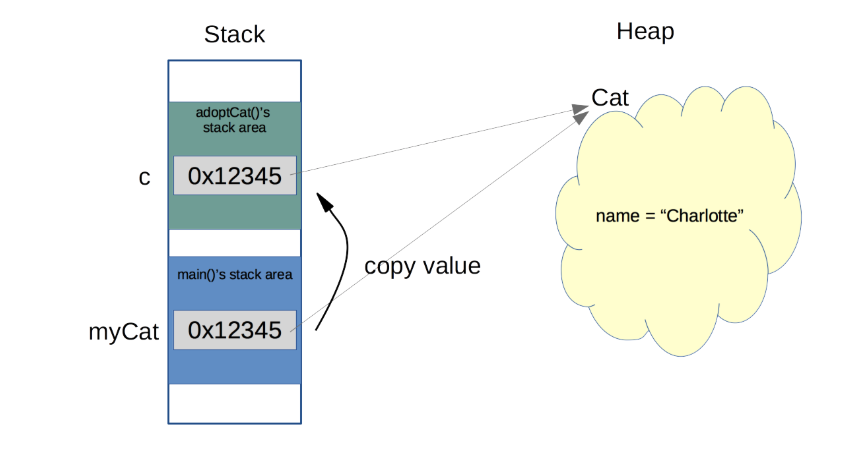


For reference types, Java passes the *reference* by value. That is, it creates a copy of the pointer to the object in memory. Let’s look back at our Cat example to see how this looks in memory:

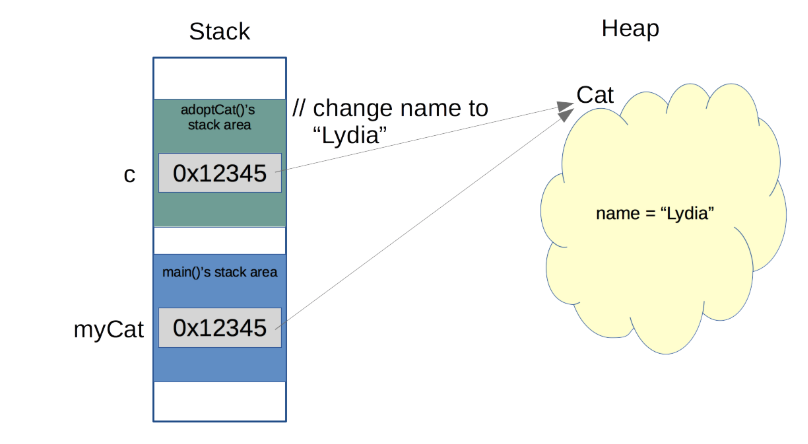
We create myCat and assign it the name “Charlotte” which creates an instance of the object on the heap and the variable myCat that refers to that particular object.



When adoptCat() is called, the parameter c gets a copy of the reference variable myCat – this is *pass-by-value*.



The method adoptCat() changes the name of the cat referred to by c (which is the *same one* as the cat referred to by myCat). This is *acting* as if we passed myCat by reference (intuitively pass-by-reference).



So, we can see that the Cat object pointed at by myCat was changed as if we had passed it by reference.

