

**COMP90041**  
Programming and Software Development  
2020 - Semester 1  
Lab 5

Yuhao(Beacon) Song  
[yuhsong1@unimelb.edu.au](mailto:yuhsong1@unimelb.edu.au)  
[yuhsong@student.unimelb.edu.au](mailto:yuhsong@student.unimelb.edu.au)

# Introduction

- ❖ **Timetable**

- ❖ Tue(18) 16:15-17:15 (Melbourne Time)
  - ❖ <https://unimelb.zoom.us/j/94854648719?pwd=WUY0NmR6MkI5UVZBUWhGNWFIU216Zz09>
- ❖ Wed(19) 17:15-18:15 (Melbourne Time)
  - ❖ <https://unimelb.zoom.us/j/93723434766?pwd=MGh4QkJuZnhJR21LZ0VqeXhJQU52UT09>

- ❖ **GitHub Page (Tutorial Materials, Solutions, Additional Resources)**

- ❖ <https://github.com/Beaconsyh08/COMP90041-2020SEM2>

- ❖ **PollEv**

- ❖ <https://pollev.com/yuhsong>

# Outline

- ❖ Assignment 1
- ❖ Reading Feedback
- ❖ Lecture Review & Demo
- ❖ Java Mechanism (Optional)

# Static Methods

- ❖ In Java, a **static** method is a method that **belongs to** a **class** rather than an **instance/object** of a class.
  - ❖ *public static returnType myMethod(parameters){ . . . }*
- ❖ **Static** methods are **invoked** using the **class name** to **replace calling object** normally  
*returnValue = MyClass.myMethod(arguments);*
  - ❖ **Static** Methods: *Math.max(); Math.round();*
  - ❖ **Non-static** Methods: **create object “keyboard” → keyborad.nextInt();**
- ❖ A **static** method could not access instance variable or **non-static** method
- ❖ However, A **static** method **can** access static variable or **another static** method

# Static Variables

- ❖ Belongs to the **class**, not to **object**
  - ❖ There is only **one copy** of a **static variable per class**,
  - ❖ unlike **instance variables** where each object has its **own copy**
- ❖ **All objects** of the class can **read and change** a **static** variable
- ❖ A static variable is declared like an instance variable, with the modifier **static**  
*private static int myStaticVariable;*

# Memory

- ❖ A computer has two forms of memory
  - ❖ **Secondary memory** is used to hold files for "permanent" storage – USB, Disk
  - ❖ **Main memory** is used by a computer when it is **running a program** - program's
- ❖ **Main memory** consists of a long list of numbered locations called **bytes**
  - ❖ Each **byte** contains **eight bits**: eight 0 or 1 digits

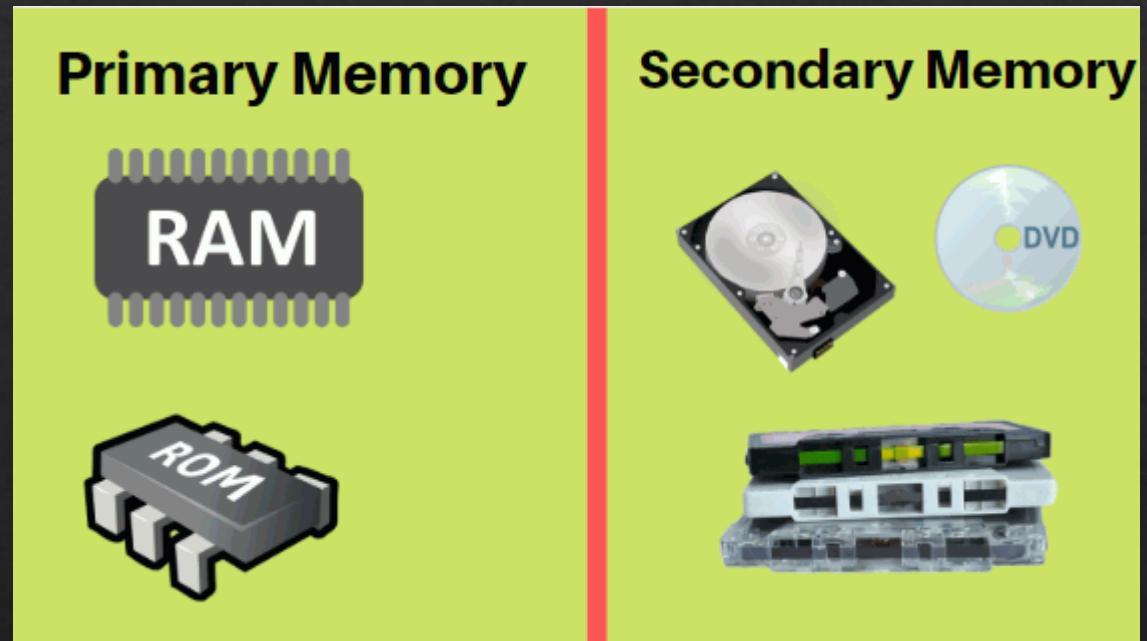
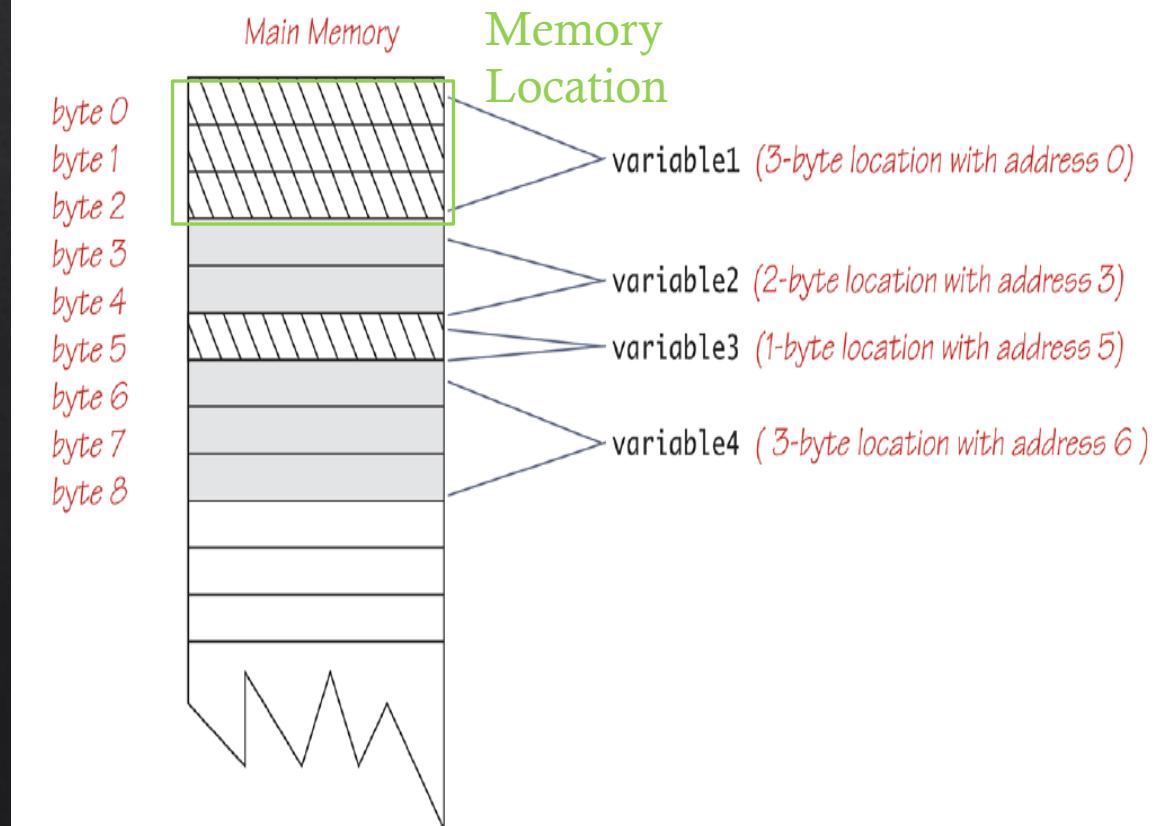


Image from:  
<https://tellghana.net/2020/01/09/differences-between-primary-and-secondary-storage-memories-devices/>

# Main Memory

- ❖ The number that identifies a byte is called its **address**
  - ❖ A data item can be stored in **one (or more)** of these bytes
  - ❖ The address of the byte is used to find the data item
- ❖ Values of most data types require more than one byte of storage
  - ❖ **Several adjacent bytes** are then used to hold the data item. The entire chunk of memory that holds the data is called its **memory location**
  - ❖ The address of the **first byte** of this memory location is used as the **address** for the data item
- ❖ A computer's **main memory** can be thought of as a **long list of memory locations** of varying sizes

Display 5.10 Variables in Memory



# Stack & Heap

- ❖ Java **Stack memory** is used for the execution of a thread. They contain **method-specific values** that are **short-lived** and **references** to other objects in the heap that is getting referred from the method.
- ❖ Java **Heap space** is used by **java runtime** to allocate memory to **Objects** and **JRE classes**. Whenever we create an **object**, it's always created in the **Heap** space.

# Primitive vs Class

- ❖ Primitive type → **value** of the variable → Stack
- ❖ Class type → **only** the **memory address** (or **reference**) where its object is **located** → Heap
- ❖ Reference: Memory Address!

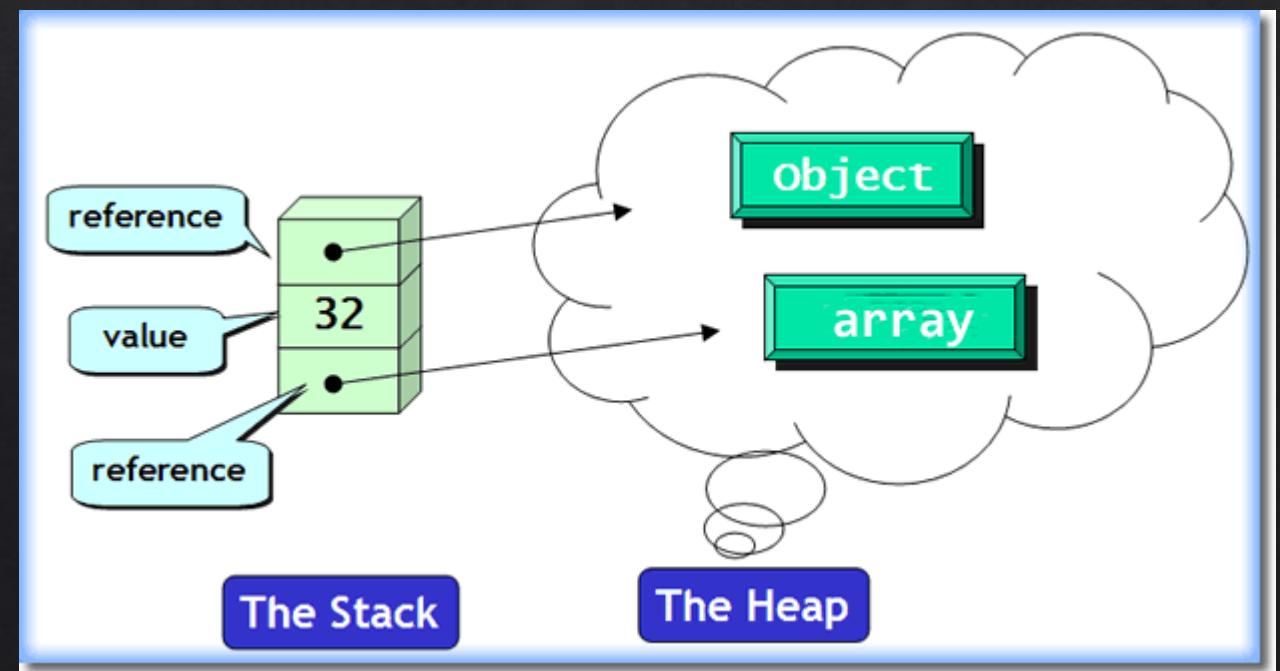


Image from:  
<https://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html>

## Display 5.12 Class Type Variables Store a Reference

---

```
public class ToyClass
{
    private String name;
    private int number;
```

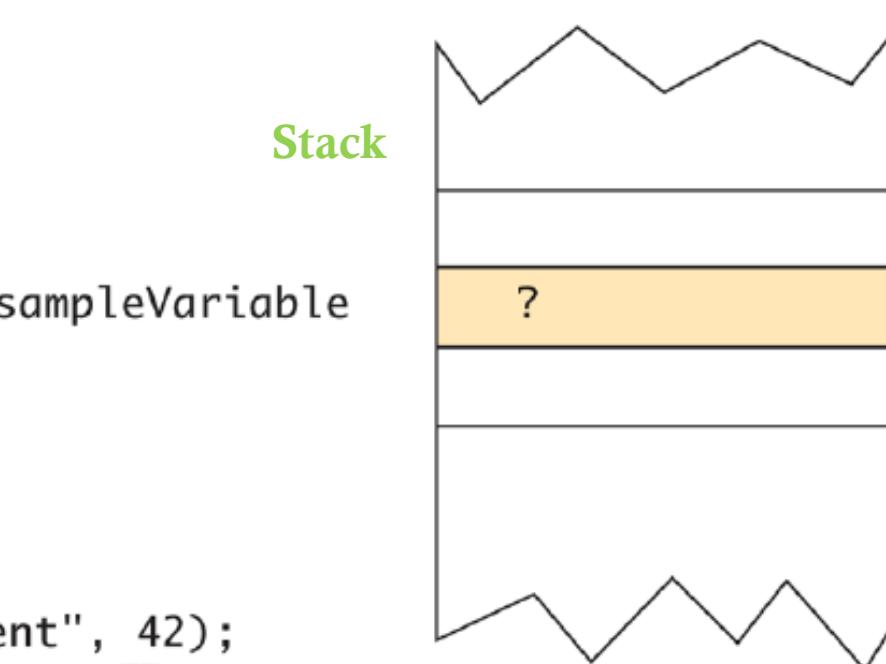
The complete definition of the class ToyClass is given in Display 5.11.

```
sampleVariable =
    new ToyClass("Josephine Student", 42);
```

Creates an object, places the object someplace in memory, and then places the address of the object in the variable sampleVariable. We do not know what the address of the object is, but let's assume it is 2056. The exact number does not matter.

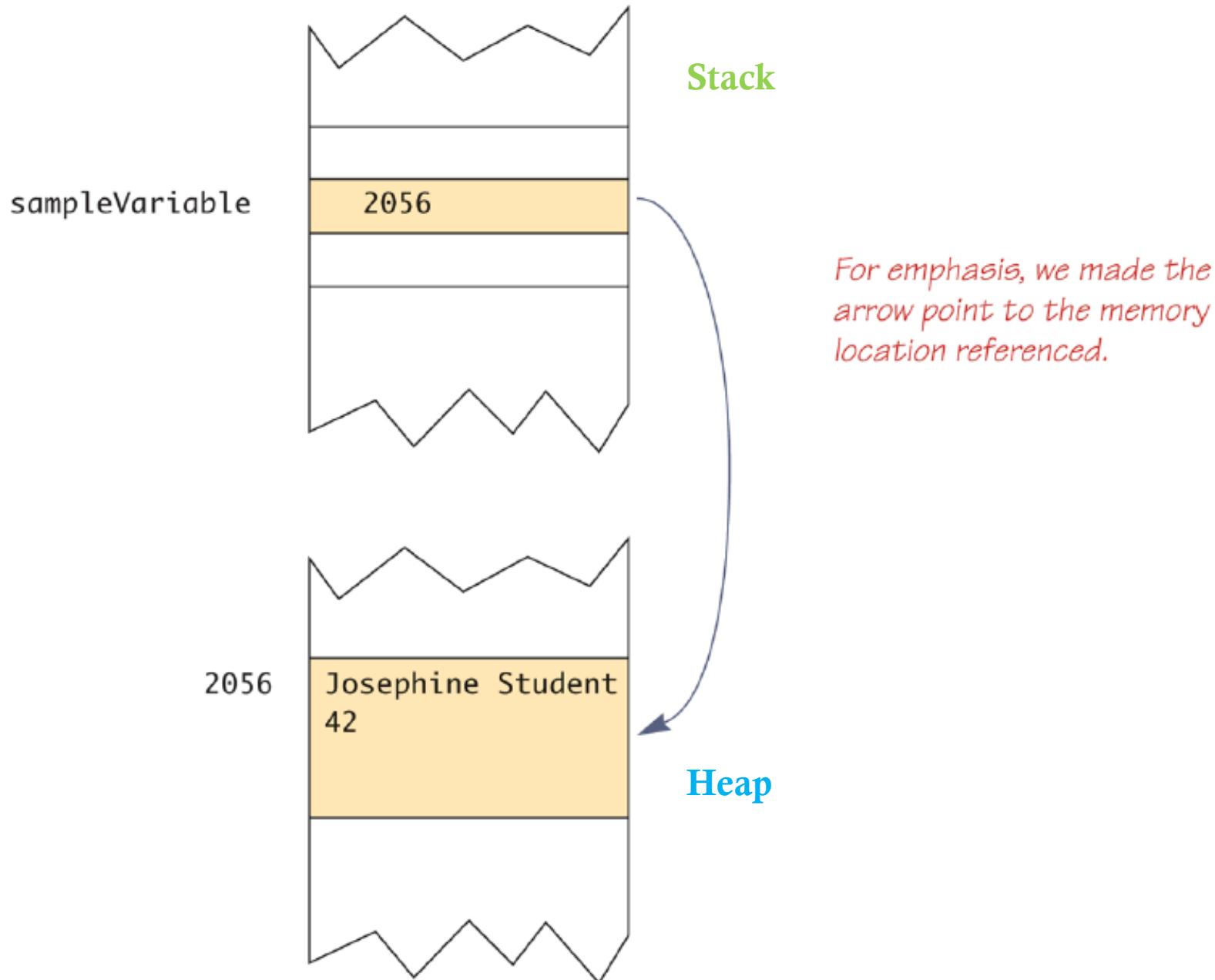
ToyClass sampleVariable;

Creates the variable sampleVariable in memory but assigns it no value.



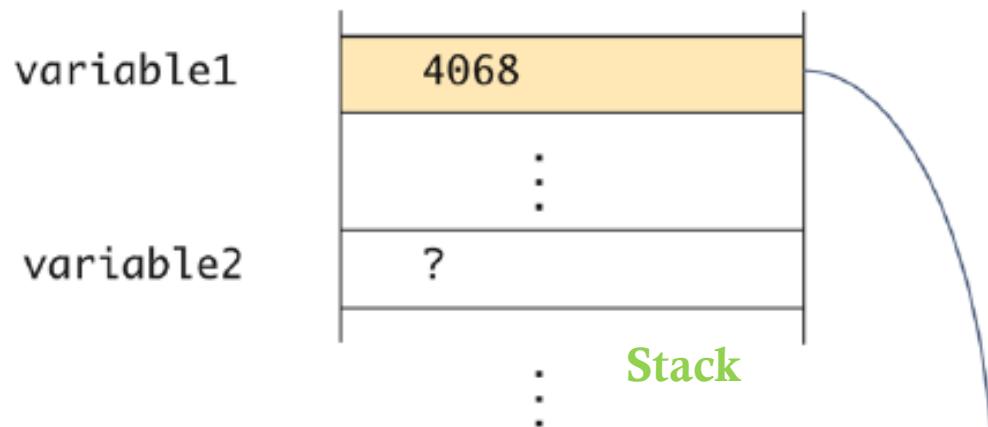
(continued)

## Display 5.12 Class Type Variables Store a Reference



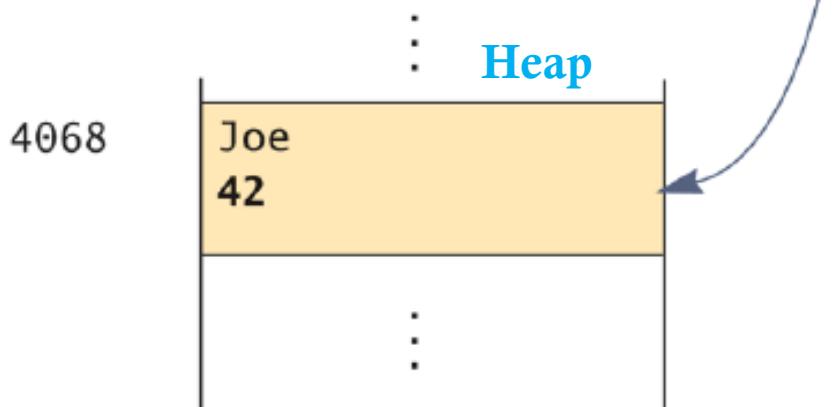
## Display 5.13 Assignment Operator with Class Type Variables

```
ToyClass variable1 = new ToyClass("Joe", 42);  
ToyClass variable2;
```



We do not know what memory address (reference) is stored in the variable **variable1**. Let's say it is 4068. The exact number does not matter.

*Someplace else in memory:*



*Note that you can think of*

*new ToyClass("Joe", 42)*

*as returning a reference.*

(continued)

```
variable2 = variable1;
```

variable1

4068

:

variable2

4068

:

Stack

refer to the same memory address

*Someplace else in memory:*

Heap

4068

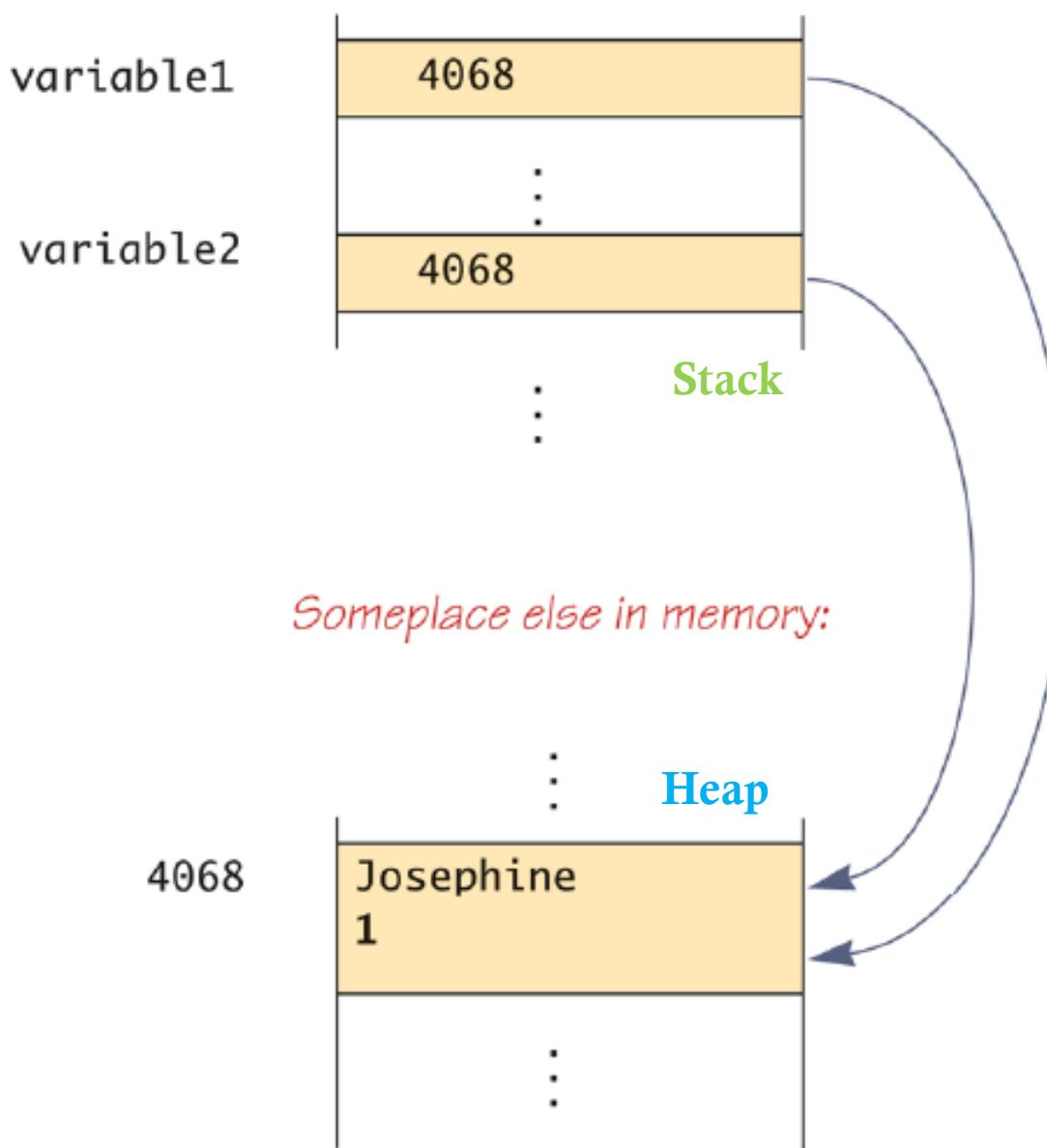
Joe

42

:

(continued)

```
variable2.set("Josephine", 1);
```



the address not change  
just the content

Notice, the value for  
**variable 1** and  
**variable 2** never  
change during the  
whole process

# Java is Strictly Pass-By-Value

- ❖ If I have a method
- ❖ Int number1 = 10;

```
public void myMethod (int number1, ToyClass toy) {number1 = 20}
```

- ❖ When I execute the method  

```
myMethod (number1, variable1);
```
- ❖ The **value** of number1, for example 10; and the **value** of variable1, which is the memory address 4068 will be passed in the method.

# What you could do

- ❖ Java is **pass by value!**

- ❖ I highly recommend that you could try to understand the mechanism being used. ☺
- ❖ It will be extremely beneficial if you consider a CS-related career in the future.
- ❖ Recommend readings:
  - ❖ <http://www.javadude.com/articles/passbyvalue.htm>
  - ❖ <https://www.javaworld.com/article/3512039/does-java-pass-by-reference-or-pass-by-value.html>
- ❖ Practice code provided. **Try it!**