COMP90041
Programming and Software Development
2020 - Semester 1
Lab 5 - Week 6

Yuhao(Beacon) Song

yuhsong1@unimelb.edu.au

Introduction

- Timetable
  - Tue(11) 14:15-15:15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/490084146
  - Tue(07) 16:15-17.15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/291505765

- Contact
  - yuhsong1@unimelb.edu.au
  - yuhsong@student.unimelb.edu.au
  - Github: https://github.com/Beaconsyh08/COMP90041_Programming_and_Software_Development_Tutorials.git

Tilman Dingler
Yesterday

Dear all, this is to confirm that **ArrayLists can be used** in Assignment 2. We had to make sure the automated test cases would not break when these were used, hence the delay. Go forth and build your arrays dynamically! :)

Edited by Tilman Dingler on Apr 27 at 12:49

(5 likes)

Outline

- Lecture Review
- Exercise & demo
- Java Mechanism
- Project B

Static Methods

- Can be used without a calling object
- A static method still belongs to a class, and its definition is given inside the class definition

```
public static returnedType myMethod(parameters)
    { . . . }
```

- Static methods are **invoked** using the class name in place of a calling object

```
returnedValue = MyClass.myMethod(arguments);
```

Math.max(); Math.round()

keyborad.nextInt()

- A static method has **no this**, so it cannot use an instance variable or method that has an implicit or explicit this for a calling object **(nonstatic)**
- A static method **can** invoke **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
- Although a **static method** cannot access an instance variable, a static method can access a **static variable**
- A static variable is declared like an instance variable, with the addition of the modifier static

```
private static int myStaticVariable;
```

Writing Java Class

1. Write a class named `Appointment` that contains instance variables `startTime`, `endTime`, and a `date` which consists of a `day`, `month` (valid values are January through December) and `year`. All times should be in military (24 hours) format, therefore it is appropriate to use integers to represent the time. Write accessor and mutator methods, and helper methods for setting and displaying an appointment.

2. Add two constructors to the `Appointment` class, including a default constructor and a constructor to initialise an Appointment to suitable arguments.

3. Write a driver program that tests your `Appointment` class.

4. Rewrite the mutator method for setting the `day` in the `Appointment` class in a way that it has to use the `this` keyword.

5. Modify mutator methods for setting `day`, `month` and `year` so that only valid values for those instance variables are accepted. Then test the class again (using your driver class) and make sure that it still works and accepts valid values only.

6. Write a precondition for the `endTime` mutator method that ensures validity of `endTime`.

7. Draw a UML class diagram for the classes used in the exercises above.

Exercise

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

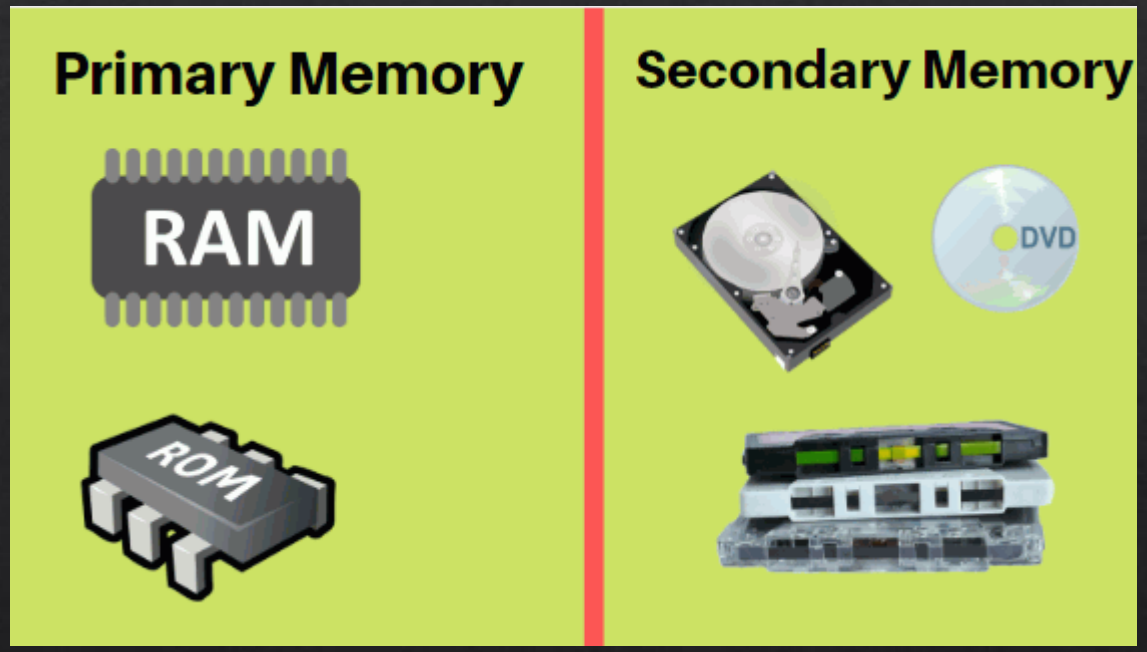
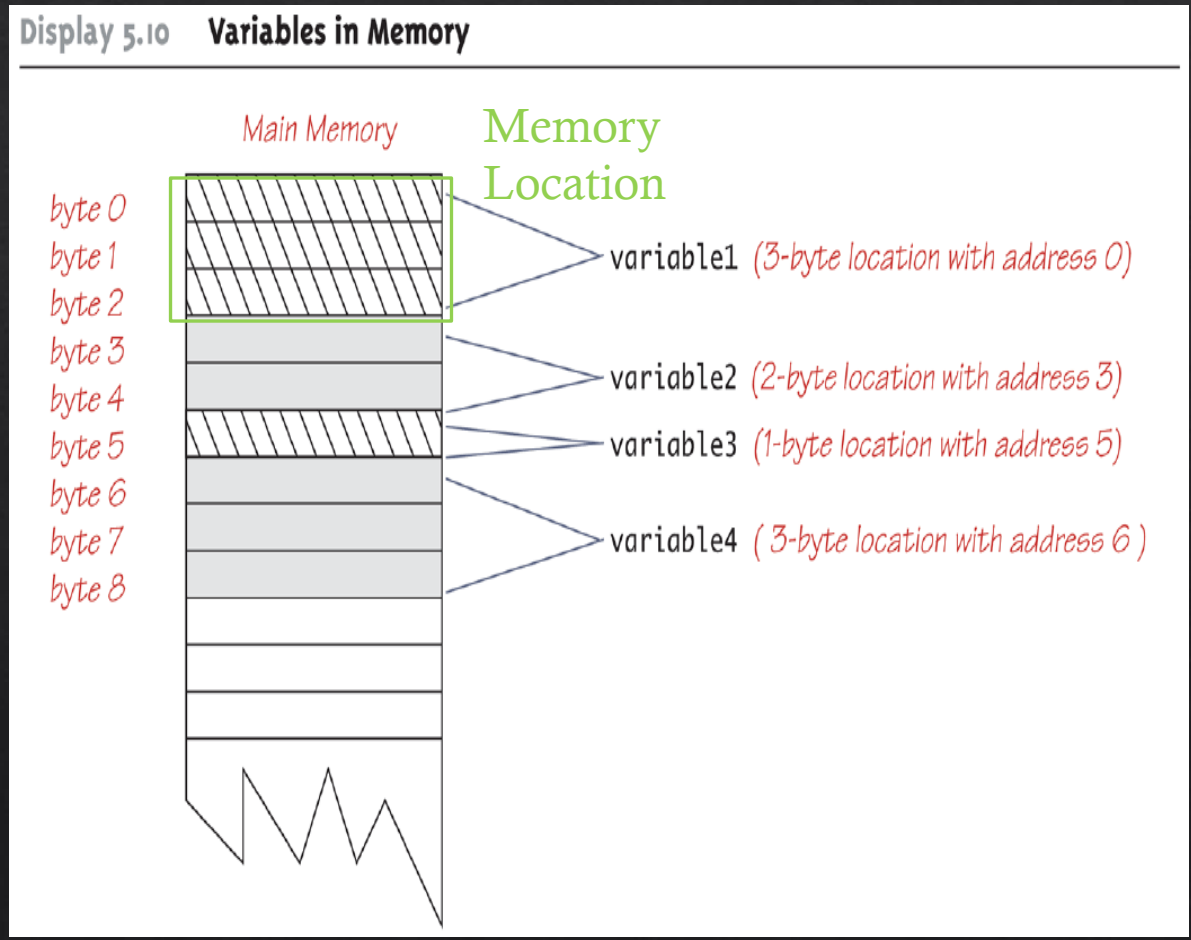


Image from: https://tellghana.net/2020/01/09/differences-between-primary-and-secondary-storage-memories-devices/

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



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.

Primitve 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!**

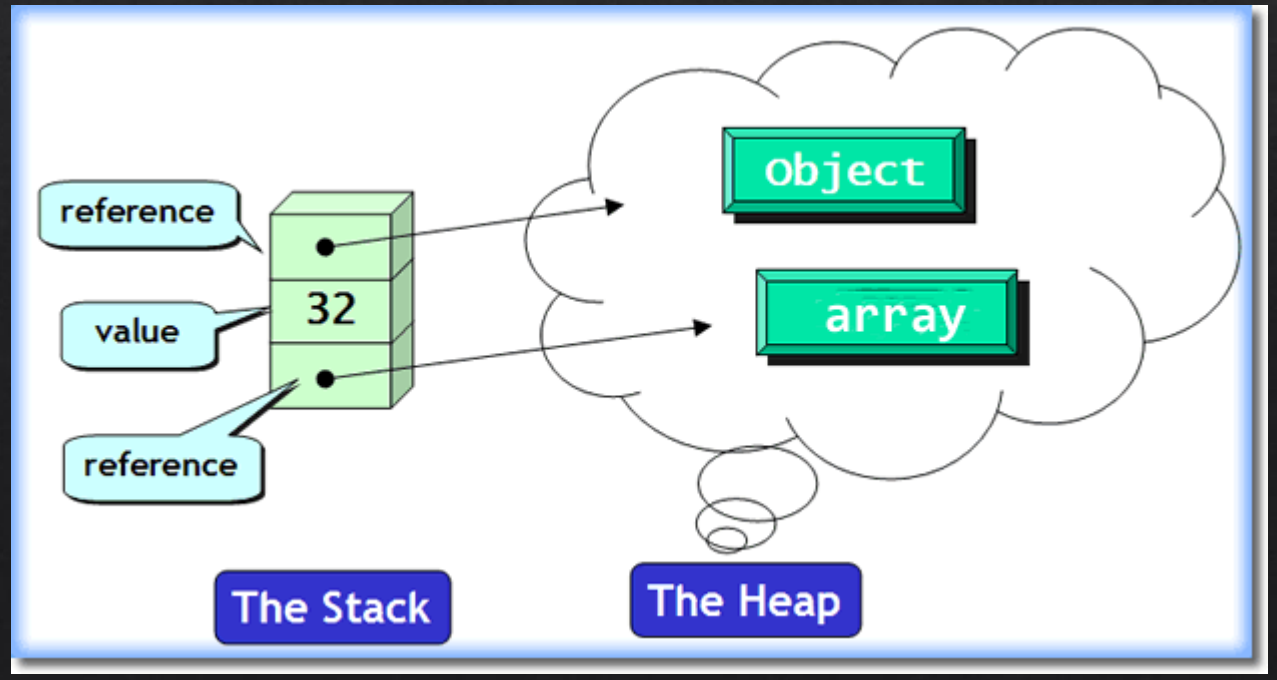


Image from:
https://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html

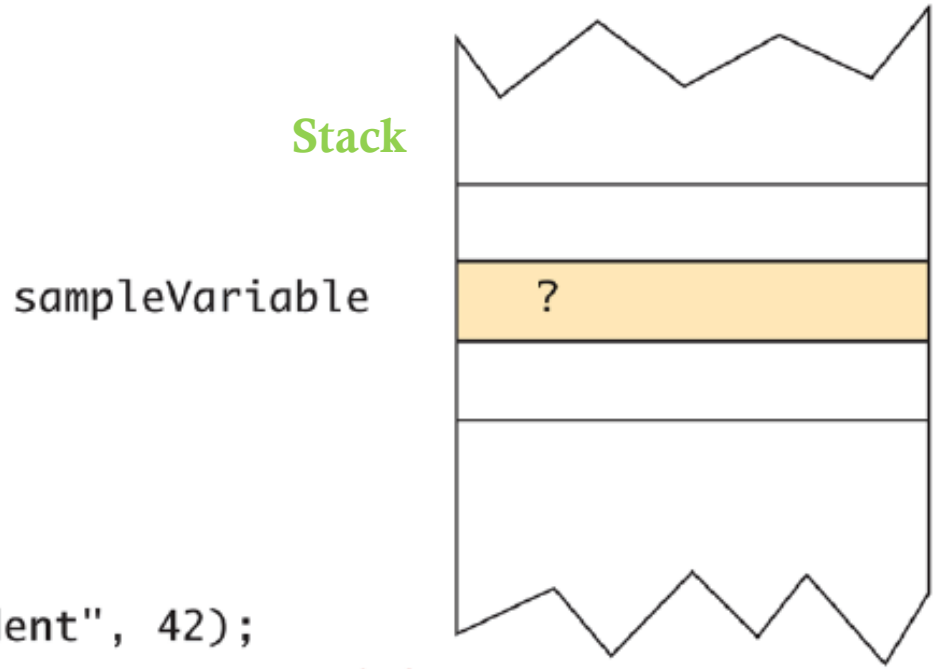
Display 5.12 Class Type Variables Store a Reference

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

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

```
ToyClass sampleVariable;
```

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

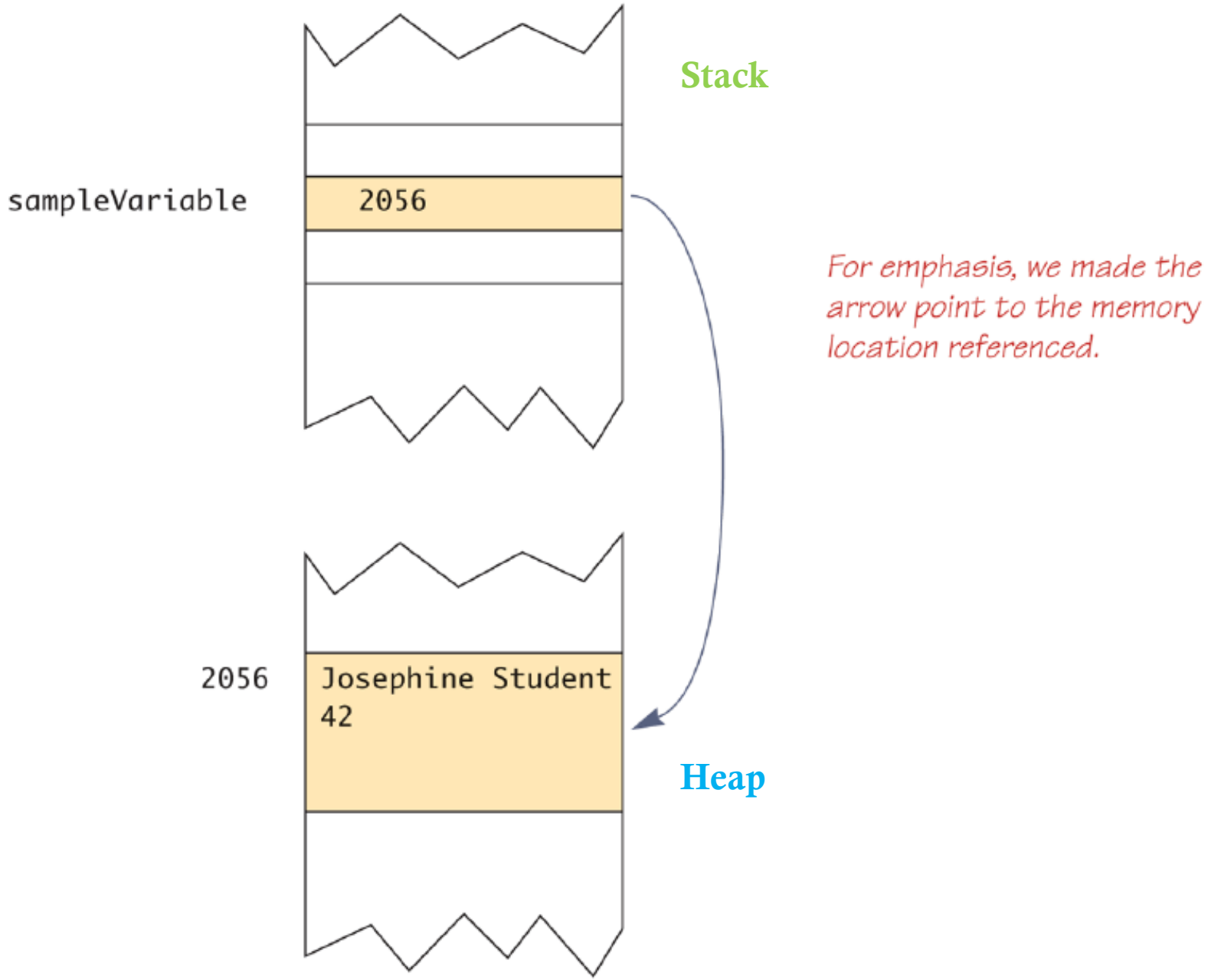



```
sampleVariable =
new ToyClass("Josephine Student", 42);
```

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.*

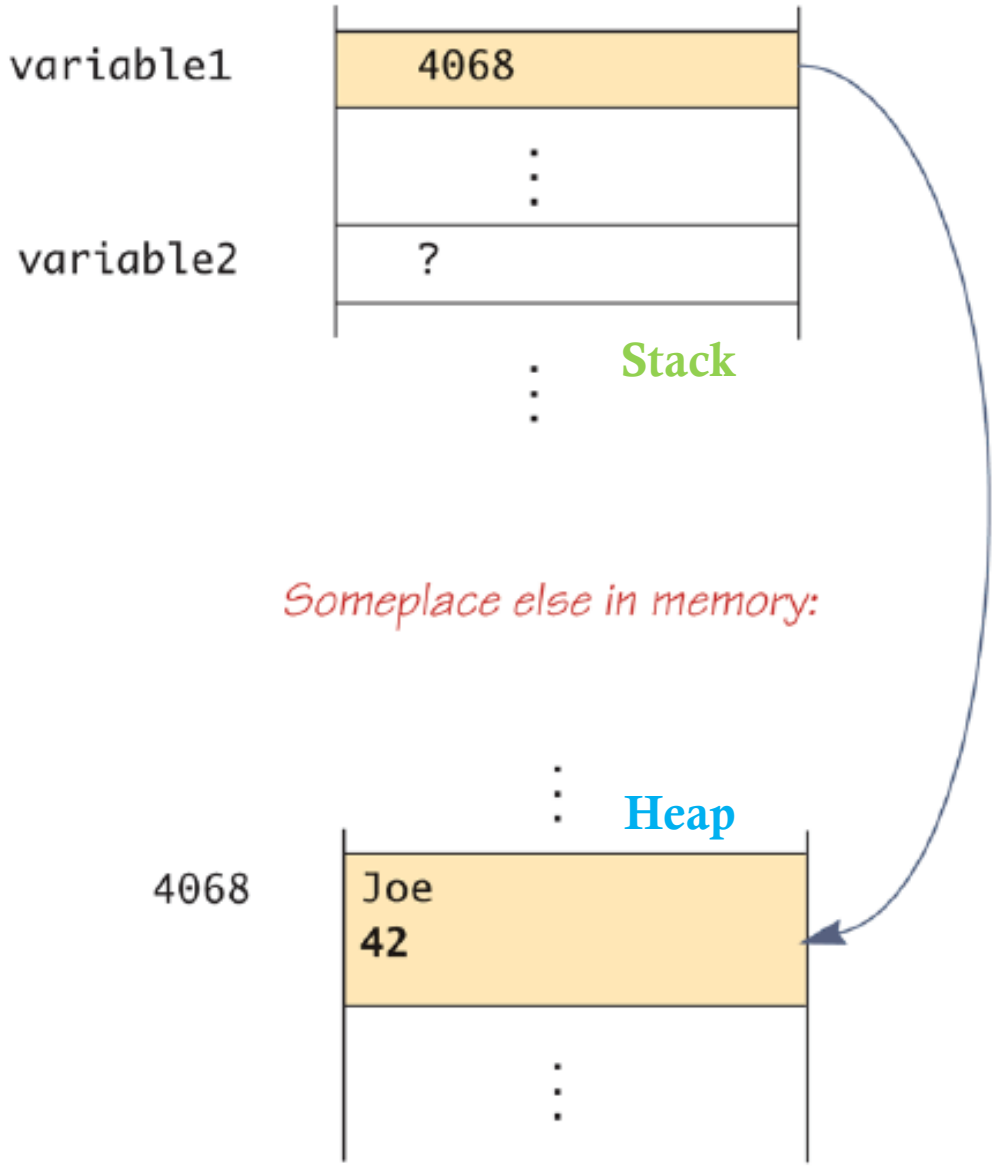
(continued)

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;
```



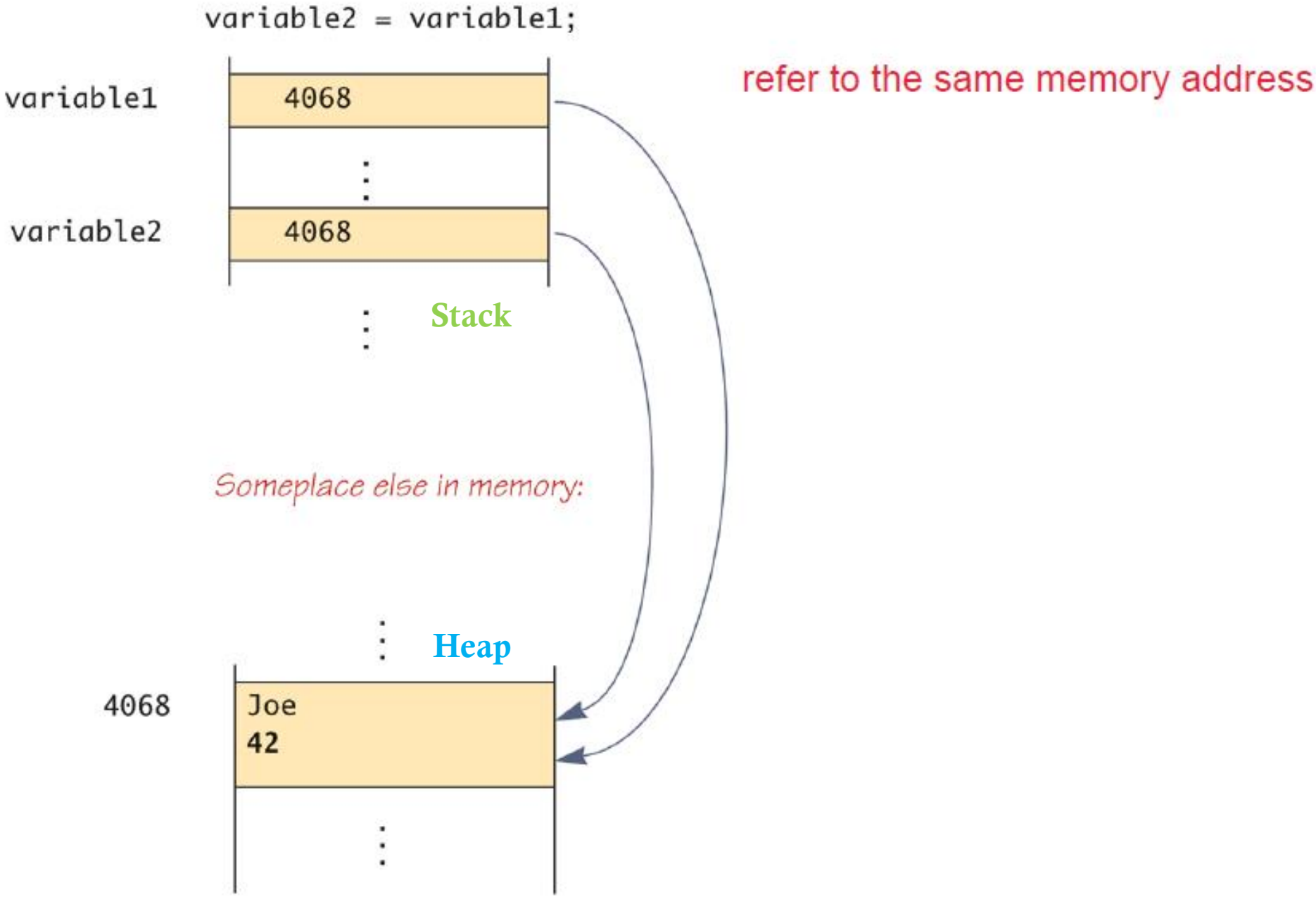
We do not know what memory address (reference) is stored in the variable `variable1`. *Let's say it is* `4068`. *The exact number does not matter.*

Note that you can think of

```
new ToyClass("Joe", 42)
```

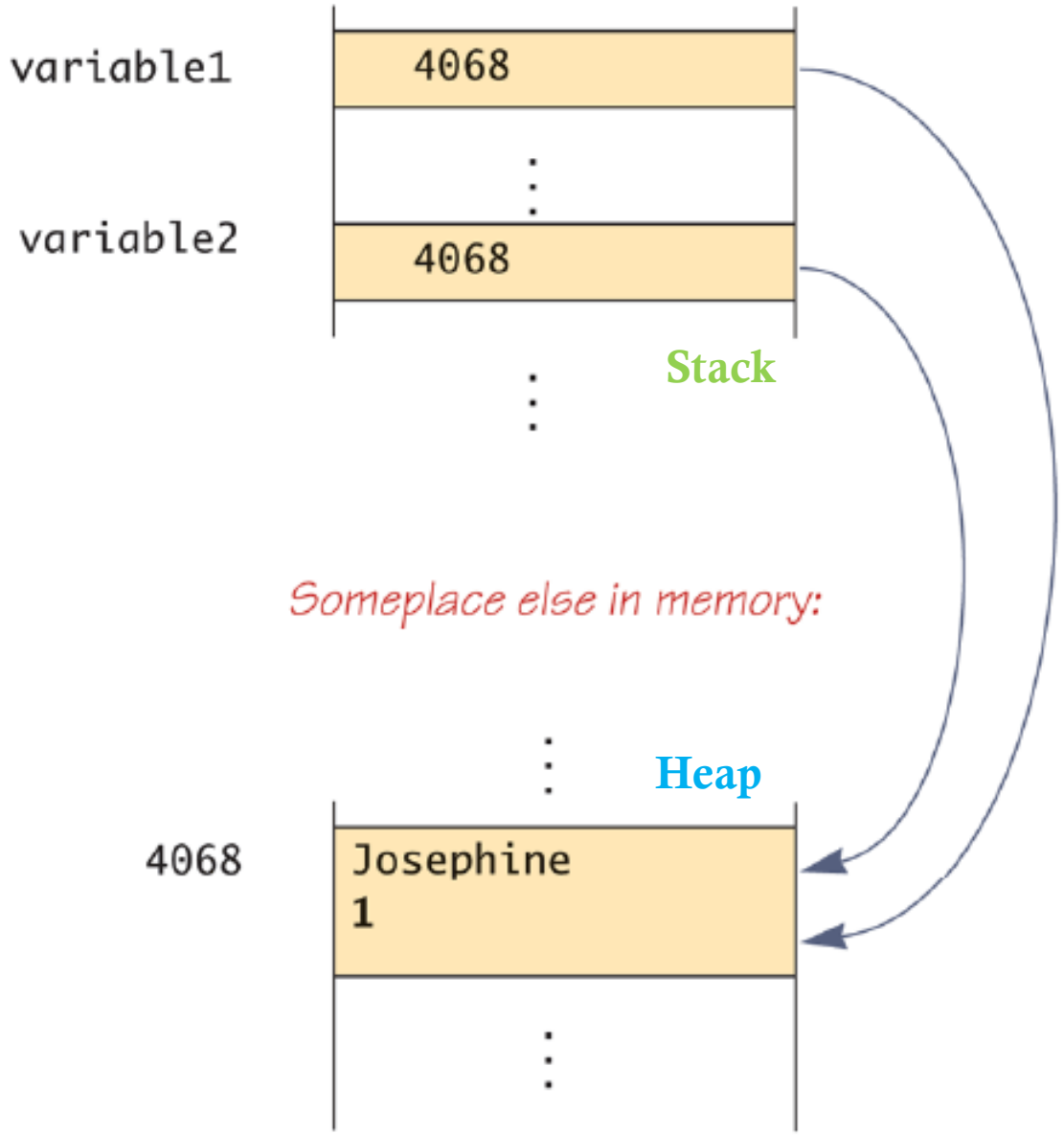
as returning a reference.

(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

  public void myMethod (int number, ToyClass toy) {…}

- 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.

Homework

- Java is **pass by value!**
- Note that it is **not graded** and **not compulsory**, but I highly recommend that you could try to understand the mechanism being used. ☺
- It will be extremely beneficial!
- 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!**