

ITI 1121. Introduction to Computing II

Data types: primitive and reference types

by

Marcel Turcotte

Version January 9, 2020

Preamble

Preamble

Overview

Data types: primitive and reference types

We examine the advantages of strongly typed languages. We compare primitive types and reference types. We introduce memory diagrams.

General objective :

- This week, you will be able to contrast primitive types and reference types.

Preamble

Learning objective

Learning objective

- ✦ **Name** predefined primitive types and references.
- ✦ **Illustrate** associations between objects using memory diagrams.

Readings:

- ✦ Pages 545-551 of E. Koffman & P. Wolfgang.

Preamble

Plan

Plan

1 Preamble

2 Theory

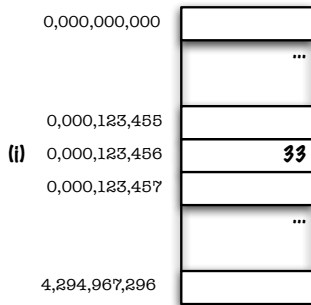
3 Prologue

Theory

Definition: Variable

What is a **variable**?

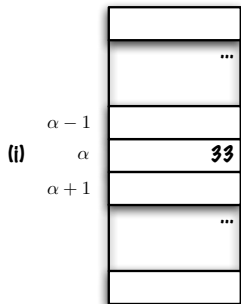
- ✚ A variable is an abstraction for a **memory location**, which is referred to using a **label** in a high-level programming language.



```
i = 33;
```

Convention

I will use **Greek letters** to designate memory (addresses) since in Java we don't know the location of objects and shouldn't worry about it.



```
i = 33;
```

Definition: Data type

What are **data types**?

- ✚ A **data type** provides information on the **representation in memory** of the data (e.g. range of possible values) as well as the **operations** that are defined for this data.

Discussion: Data types

But then again, **who** benefits from data types?

- ❖ The **compiler** to reserve the necessary memory space for the data.
- ❖ The **compiler**, but also to the **programmer**, in order to detect certain errors at compile time — applying an operation undefined for a particular data type.

Discussion: Data types

Give **examples** of data types?

- ❏ byte, short, int, long
- ❏ float, double
- ❏ boolean
- ❏ char

Predefined types

Type	Size	Maximum	Examples
boolean	1		true , false
char	16	'\uFFFF'	'a', 'A', '1', '*'
byte	8	127	-128, -1, 0, 1, 127
short	16	32767	-128, -1, 0, 1, 127
int	32	2147483647	-128, -1, 0, 1, 127
long	64	9223372036854775807	-128L, 0L, 127L
float	32	3.4028235E38	-1.0f, 0.0f, 1.0f
double	64	1.7976931348623157E308	-1.0, 0.0, 1.0

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>
- <https://docs.oracle.com/javase/specs/jls/se12/html/jls-4.html>

Java : Data types

- Java is a **strongly typed** language. Which means that every **variable** and every **expression** have a type known at the time of **compilation**.
- One must declare the **type** of each **variable** and **parameter**, as well as the type of the **return value** of the methods.

type identifier
int age ;

Compile time error: «cannot find symbol»

```
public class Test {  
    public static void main(String [] args) {  
        age = 21;  
    }  
}
```

- ✚ In the above example, the variable **age** was not declared.

```
Test.java:3: error: cannot find symbol
```

```
    age = 21;
```

```
    ^
```

```
symbol:   variable age
```

```
location: class Test
```

```
1 error
```

Solution

- One must declare the **type of the variable**, here **int** (line 3), before using it (line 4).

```
1 public class Test {  
2     public static void main(String [] args) {  
3         int age;  
4         age = 21;  
5     }  
6 }
```

Type declaration: methods

```
public type int sum(type int a , type int b) {  
    return a+b;  
}
```

- ✚ One must declare the **type** of each **parameter**, as well as the type of the **return value** of the methods.

Compile time error: return value and parameters

```
public class Test {  
    public sum(a, b) {  
        return a+b;  
    }  
}
```

Test.java:2: error: invalid method declaration; return type required

```
public sum(a, b) {  
    ^
```

Test.java:2: error: <identifier> expected

```
public sum(a, b) {  
    ^
```

Test.java:2: error: <identifier> expected

```
public sum(a, b) {  
    ^
```

3 errors

Type of the return value: void

- Some methods **do not return any result**, this is the case of the method **swap** below, the type of the return value is then **void** (“returns nothing”).

```
public static void swap(int [] xs) {  
    int tmp;  
    tmp = xs[0];  
    xs[0] = xs[1];  
    xs[1] = tmp;  
}
```

Type of the return value (compile time error)

```
public class Test {  
    public static swap(int [] xs) {  
        int tmp;  
        tmp = xs[0];  
        xs[0] = xs[1];  
        xs[1] = tmp;  
    }  
}
```

```
> javac Test.java
```

```
Test.java:2: error: invalid method declaration; return type required
```

```
    public static swap(int [] xs) {  
        ^
```

```
1 error
```

Type and assignment (compile time error)

```
public class Test {  
    public static void testTypes() {  
        boolean b;  
        b = "true";  
    }  
}
```

```
> javac Test.java
```

```
Test.java:4: error: incompatible types: String cannot be converted to boolean
```

```
    b = "true";
```

```
        ^
```

```
1 error
```

Solution

```
public class Test {  
    public static void testTypes() {  
        boolean b;  
        b = true;  
    }  
}
```


Types and expressions

```
public class Test {  
    public static void testTypes() {  
        if (3 < 4 && 0) {  
            System.out.println("Bingo!");  
        }  
    }  
}
```

```
> javac Test.java
```

```
Test.java:3: error: bad operand types for binary operator '&&'
```

```
    if (3 < 4 && 0) {  
           ^
```

```
        first type: boolean
```

```
        second type: int
```

```
1 error
```

Solution

```
public class Test {  
    public static void testTypes() {  
        if (3 < 4 && 'a' == 'a') {  
            System.out.println("Bingo!");  
        }  
    }  
}
```

Java: Data types (continued)

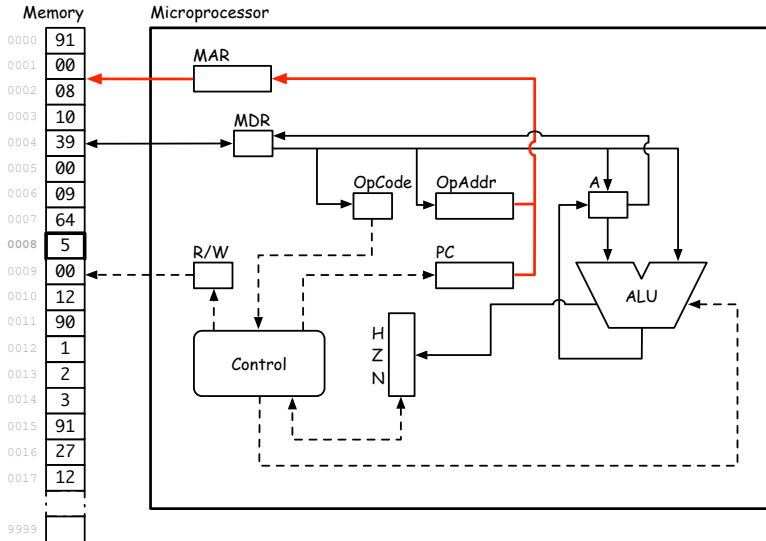
- ✦ There are **primitive types** and **reference types**.
- ✦ What is a **primitive type**? What is a **reference type**?

Java: data types (continued)

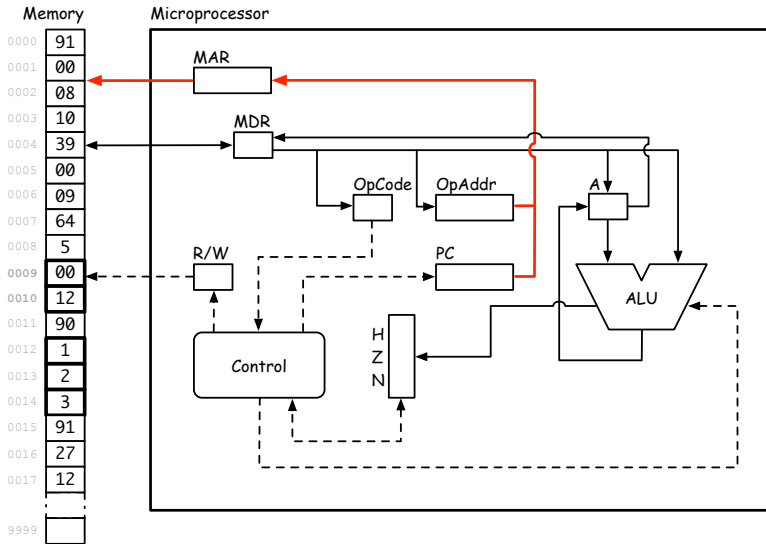
- ❖ The **primitive types** are:
 - ❖ numbers (**byte**, **int**, **long**, **float**, **double**), characters (**char**, but not the strings) and boolanns (**booleans**)
 - ❖ **the value of a variable of a primitive type is found at the address designated by the label (identifier).**
- ❖ **References:**
 - ❖ Predefined:
 - ❖ Arrays
 - ❖ Strings
 - ❖ Types defined by the user, references to objects.
 - ❖ **The value of a reference type variable is the address of the memory location of the object designated by the variable — it is said that the variable points, designates or references the object.**

Primitive vs reference and the TC-1101

```
int pos;  
pos = 5;  
  
int [] xs;  
xs = new int [] {1, 2, 3};
```

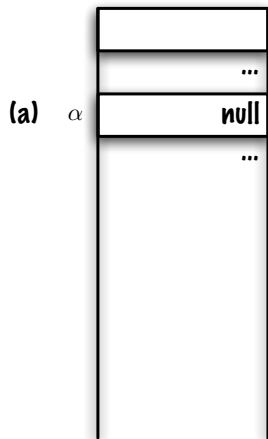


- The variable **pos** is of type **int**, a primitive type, if **pos** designates the address **00 08**, then the value **5** is saved at the address **00 08**.



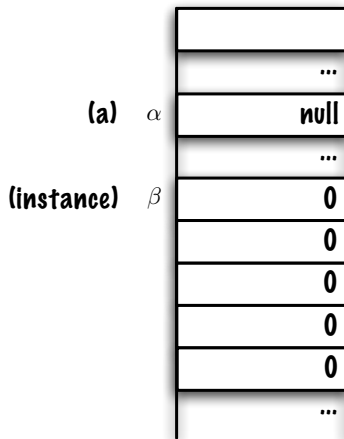
- + The variable **xs** is of type **reference** to an array of integers, if **xs** designates the address **00 09**, then the value of the cells **00 09** and **00 10**, is the address where the array was saved in memory, **00 12**. At address **00 12** is the array, with its three values **1**, **2**, and **3**.

```
> int [] a;  
a = new int [5];
```



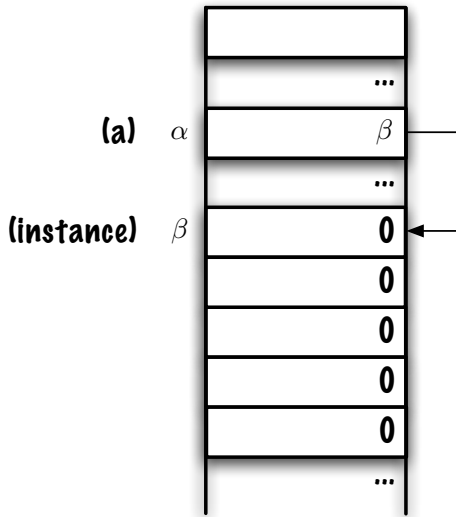
The declaration of a variable of type reference **does not create the object (instance)**, the compiler will reserve enough space to contain the reference (pointer), **null** is a literal which means: does not designate any object.


```
int [] a;  
> a = new int [ 5 ];
```



The creation of an object, **new int[5]**, reserves a portion of memory for 5 integers (and for internal management — *housekeeping*). Each cell in the array behaves as a variable of type **int** and receives the initial value 0.

```
int a [];  
> a = new int [ 5 ];
```



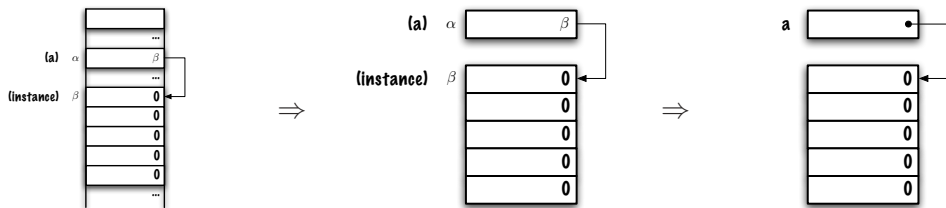
Finally, the reference of the new object is saved at the address designated by the label **a**.

Theory

Memory representation

Memory diagrams

Since we don't know the location of objects in memory (and shouldn't worry about it), we'll use memory diagrams (rightmost image).



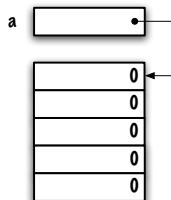
Memory diagram

Rules for your memory diagrams:

- One box for each **reference** variable and one **arrow** to the designated object.
- A box for each variable of type **primitive** type and its **value** in the box itself.

```
int [] a;  
a = new int [ 5 ];
```

⇒



Prologue




Summary

- ❖ A variable is an abstraction for a **memory location** that is referred to using a **label**.
- ❖ A **data type** provides information on the **representation in memory** of the data (e.g. range of possible values) as well as the **operations** that are defined for this data.
- ❖ The value of a variable of a **primitive type** is found **at the address designated by the label** (identifier).
- ❖ The value of a variable of **reference type** is the address of the memory location of the object designated by the variable.
- ❖ In Java, you have to declare the type of variables.

Next module

- **Data types** (part 2)

References I

-  E. B. Koffman and Wolfgang P. A. T.
Data Structures: Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.
-  D. J. Barnes and M. Kölling.
Objects First with Java: A Practical Introduction Using BlueJ.
Prentice Hall, 4e edition, 2009.
-  P. Sestoft.
Java Precisely.
The MIT Press, second edition edition, August 2005.



Marcel Turcotte

Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EECS)
University of Ottawa