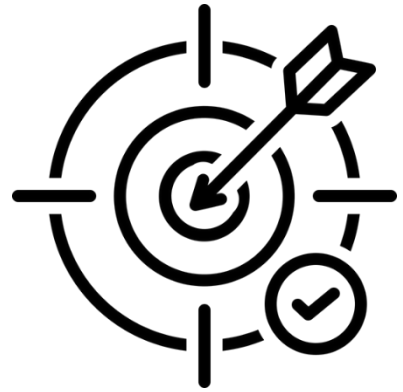
Variable & Constant

Object-Oriented Programming

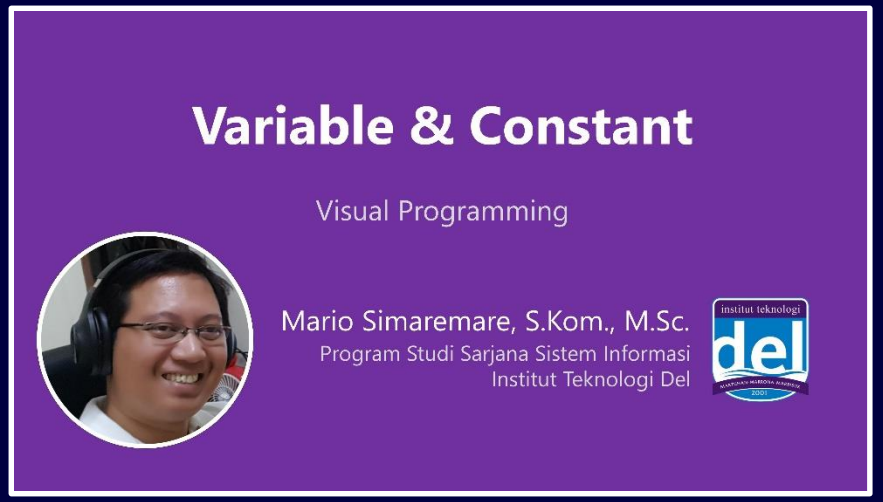
Mario Simaremare, S.Kom., M.Sc.
Program Studi Sarjana Sistem Informasi
Institut Teknologi Del

Objectives

- The objectives of this session are the following:
  - The students are able to elaborate the basic concept of variable and constant.
  - The students are able to elaborate the difference between primitive and object-reference variables.
  - The students are able declare and initialize variables and constants.
  - The students are able to describe the concept of immutable object.

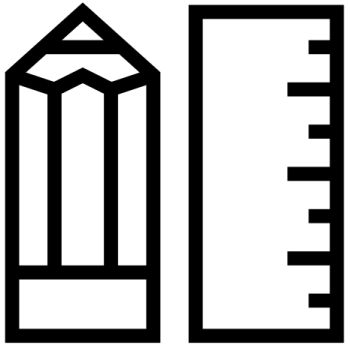


Please see these materials first



Outlines

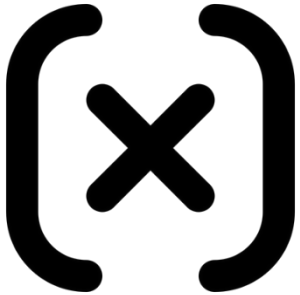
1. The basic concept of variable.
2. Variable vs. constant.
3. Primitive-typed vs. object-reference variable.
4. Declaring and initializing variables.
5. Immutable objects.



The Basic Concept of Variable

Variable

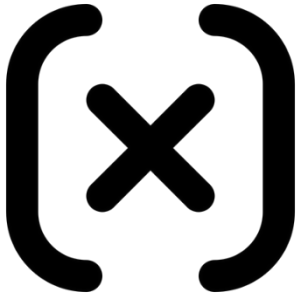
- A variable is a named location in memory that is able to hold a value at a time.
  - The value is **modifiable** during the program execution.
  - The value must be compatible to the variable type.
  - The name (identifier) must be unique within the scope.



- In Java, keep in mind that:
  - A variable must be declared before using it.
  - Uninitialized variable may cause an exception.
    - Hence, always initialize a brand new variable with a default value.

Variable identifier

- A variable must be identified with a unique identifier.
  - The uniqueness is limited within the variable scope.

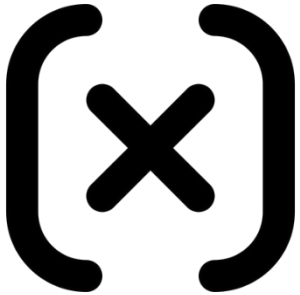


- In Java, identifier naming convention:
  - Variable identifier should be short yet meaningful, noun.
  - Only starts with an underscore or a letter.
    - Followed by any combination of underscore, letter, and number.
  - Camel-case for multiple-words identifier.
    - E.g. `fullName`, `studentId1` , `message`, etc.

Variable vs. Constant

Constant

- A constant is a **non-modifiable** variable.
  - Also called as read only variable.
  - It is set once, and will stay that way.



- Some rules to define a constant:
  - Marked with `final` keyword.
  - Snake-case and all-caps identifiers.
    - `MAX_COUNT`, `DEFAULT_SIZE`. etc.

Constant

```
package example.hello;

public class HelloWorld {

    private static final String DEFAULT_GREETINGS = "Hallo, ";

    public static void main(String[] _args) {
        StringBuilder builder = new StringBuilder(DEFAULT_GREETINGS);

        final int i = 100;
        i++;
        builder.append(i);
        builder.append("!");

        System.out.println(builder.toString());
    }

}
```

A constant of object-reference, `String`.

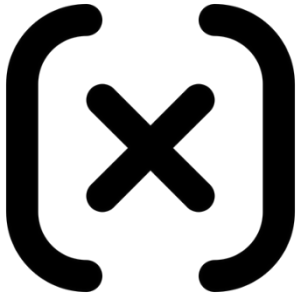
A constant of a primitive-typed, `int`.

Will surely fail

Primitive-typed Variable vs. Object-reference Variable

Variable types

- Java supports two types of variable:
  - **Primitive-typed** variable.
    - Hold a value of a specific primitive type.
    - We have discuss this.
  - **Object-reference** variable.
    - Hold a reference that points to an object (or instance).

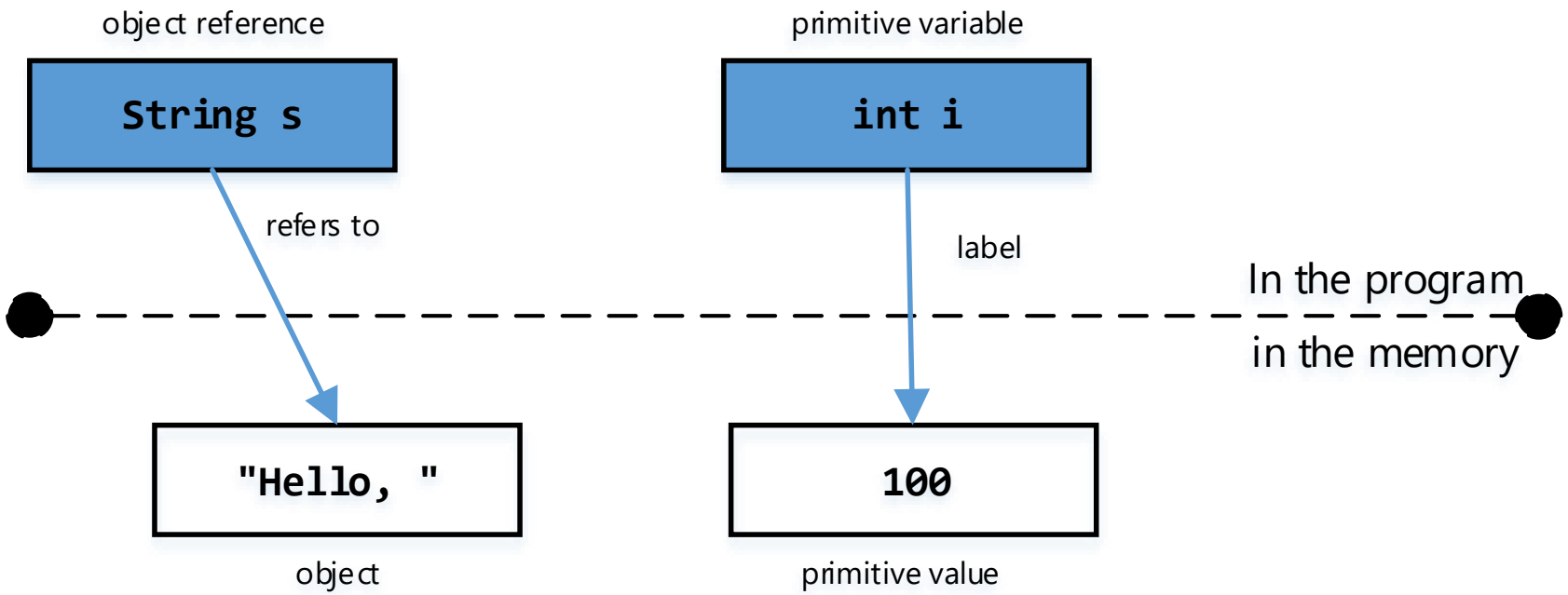


Object-reference types

- An object is a physical instance of a class.
  - It leaves in the physical memory heap.
  - The program interacts with objects via object-references.
    - An object reference might in the form of either **class**, **interface**, or **enumeration** types.
    - Multiple references may point to the same object.
    - The object will be marked for garbage collection if there is no object reference point to the object.
  - The new keyword is used to instantiate a new object.


```
// List is an interface, but ArrayList is a concrete class.
// The ArrayList is one of the List's implementation.
List<String> lString = new ArrayList<String>();
```

Primitive vs. object-reference



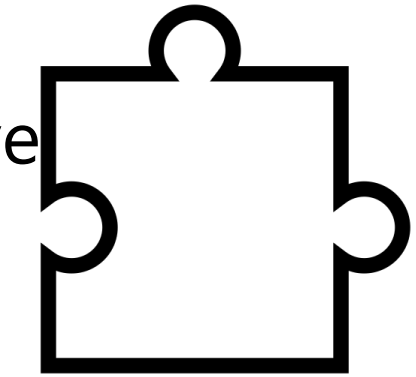

```
String s = "Hello, ";
int i = 100;
```

Declaring and Initializing Variables

Declaring a primitive-typed variable

- A variable must be declared before it is available.

- Declaration is telling the underlying platform to reserve a fraction of memory space.
  - The space length is depends on the data type length.
    - E.g. a `char` will only require 2 bytes, an `int` will require 4 bytes, etc.



```
// type identifier1 [ = value ][, identifier2 [= value ] ...];
int height;
boolean isBlue;
```

Initializing a primitive-typed variable

- Initialization is setting up a variable with a default value.

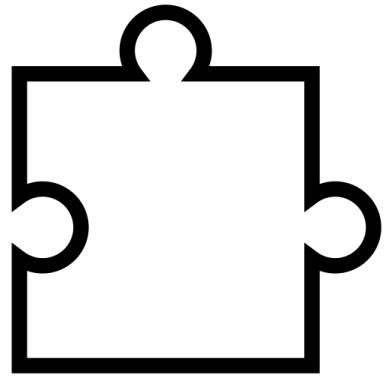
- It is a good practice to avoid exception due to uninitialized variable.
  - At runtime, Java will try to initialize variables which are left uninitialized with default values.
    - Still it would be better if the initializations are made in an explicit fashion.

Default value
Integers ← `0`
Floating-points ← `0.0`
Char ← `'\u0000'`
boolean ← `false`

```
// type identifier1 [ = value ][, identifier2 [= value ] ...];
int height = 0;
boolean isBlue = true;
```

Declaring an object-reference variable

- Declaring an object reference is very similar to declaring a primitive typed variable.



- As the name suggests, an object-reference refers to an object or an instance a class.
  - However, the object-reference itself can be in the form of either class, abstract class, interface, enumeration or other construct.

```
// List is an interface
List<String> lString;
```

Initializing an object-reference variable

- The same spirit of initialization also applies here.
  - The key difference is the severity caused by uninitialized object-reference is higher.
    - It may raise an exception that stops the execution.

Default value
Object reference ← `null`

- It is highly encouraged to always instantiate a new object for the object-reference.
  - To instantiate a new object, use the `new` keyword.

```
// Type identifier1 = new Type([arg[, arg ...]]);
// The ArrayList is one of the List's implementation.
List<String> lString = new ArrayList<String>();
```

Immutable Object

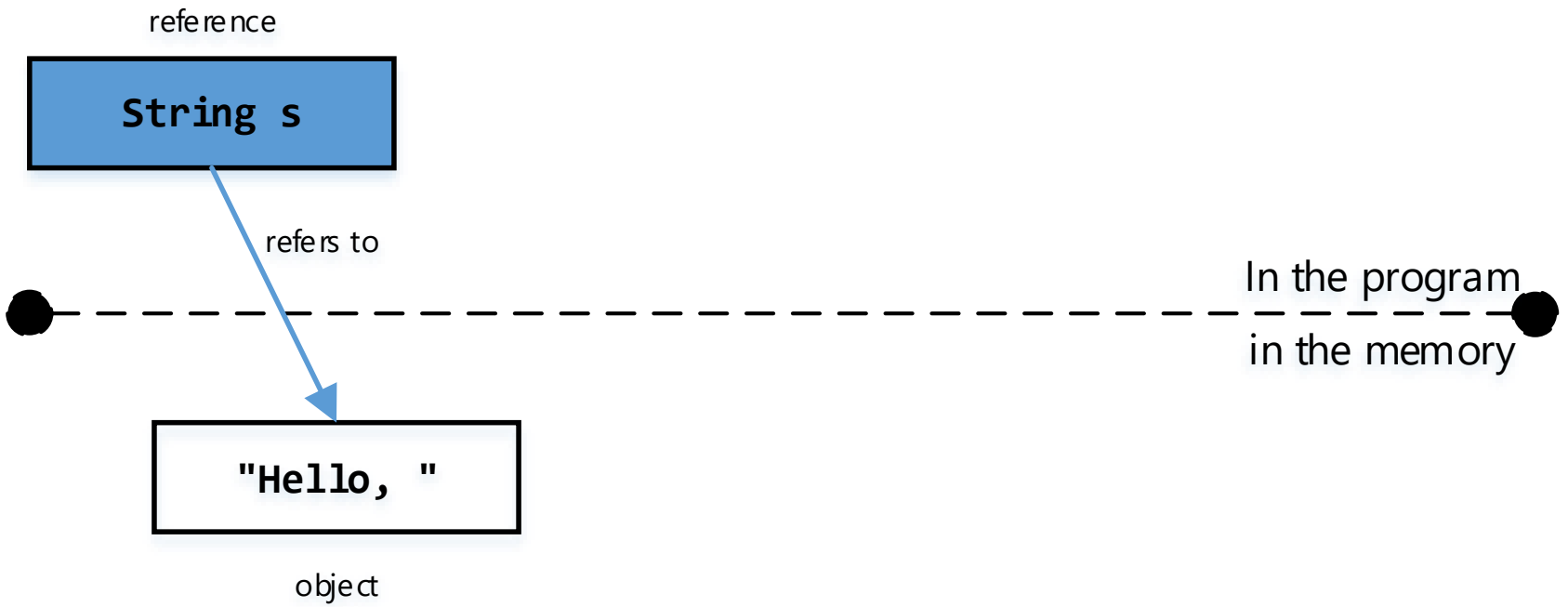
Immutable object

- Immutable objects are **non-modifiable** objects.
  - Classes that generate immutable objects:
    - `java.lang.String`, wrapper classes, user-defined immutable classes.

- What if an action is performed and will modify it's state?
  - A new object that represents the modified state is created.
  - The reference then moved to point to the new object.

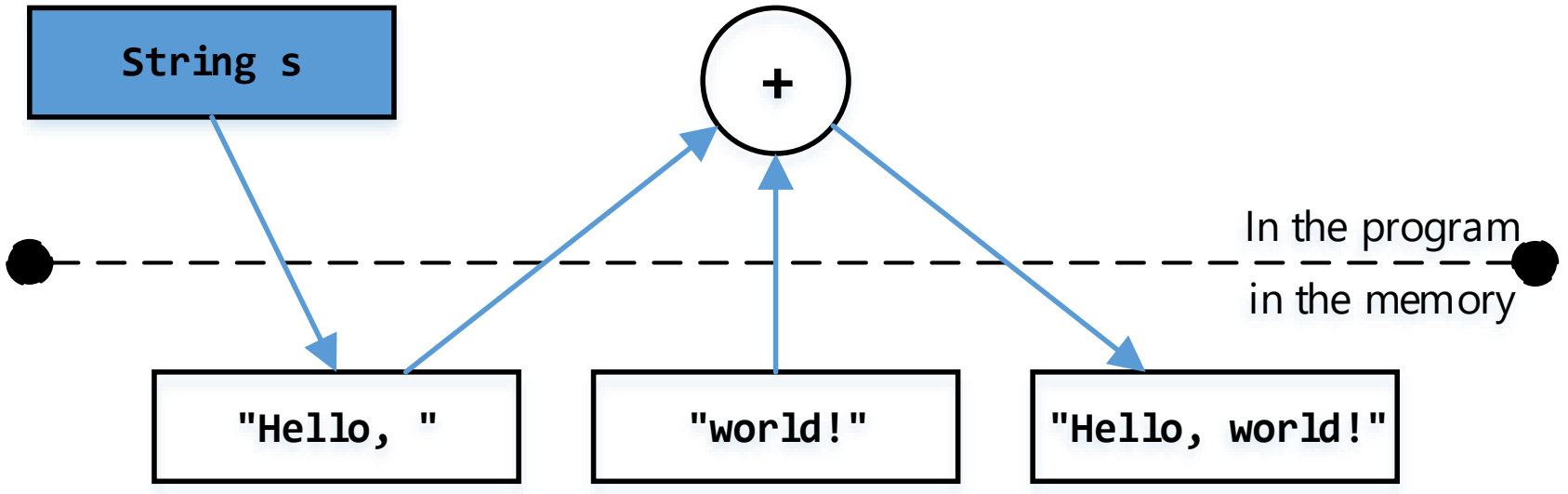
```
String s = "Hello, ";
s += "world!";
```

Immutable object



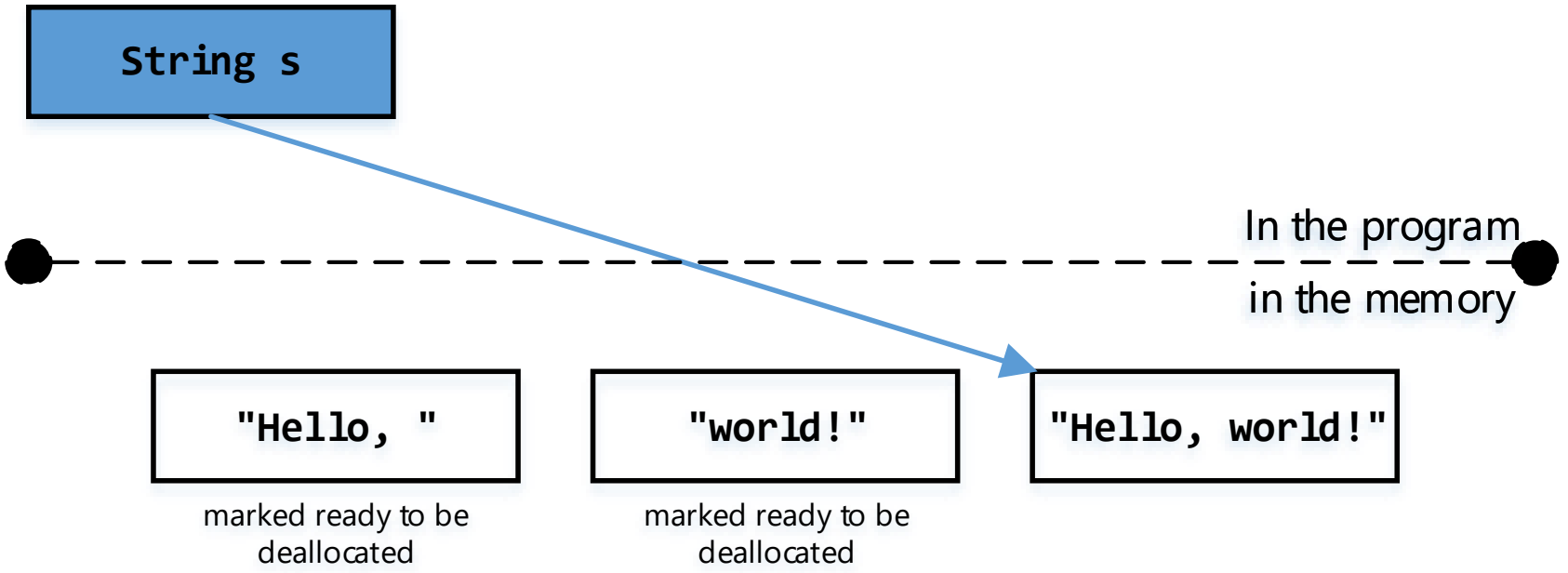

```
String s = "Hello, ";
s += "world!";
```

Immutable object




```
String s = "Hello, ";
s += "world!";
```

Immutable object




```
String s = "Hello, ";
s += "world!";
```

References

- Cay Horstman. Core Java.
- Matt Weisfeld. The Object-Oriented Thought Process.
- The Java Tutorial: Immutable Objects https://docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html

Thank you