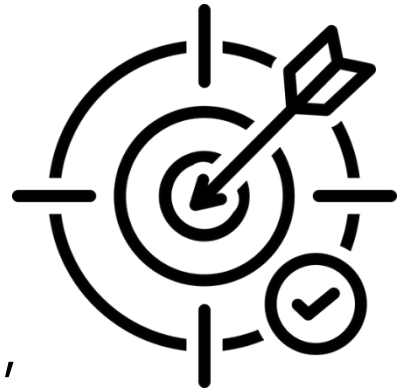
Data & Data Types

Object-Oriented Programming

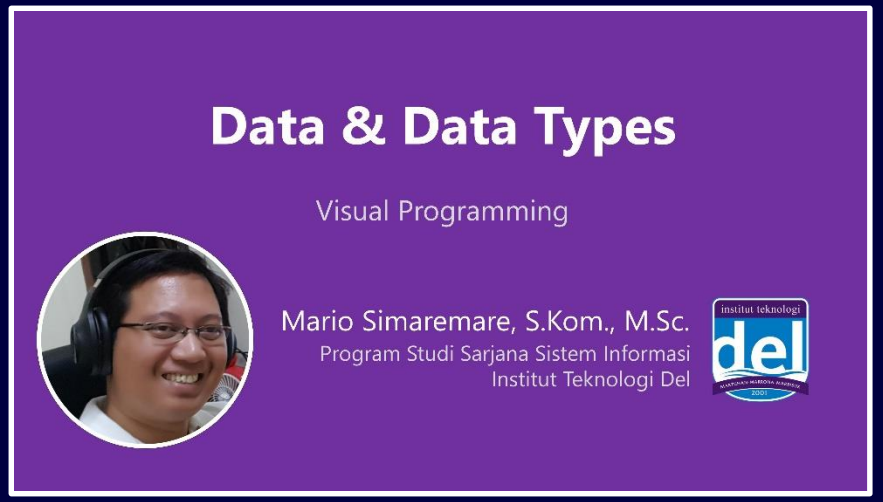
Mario Simaremare, S.Kom., M.Sc.
Program Studi Sarjana Sistem Informasi
Institut Teknologi Del

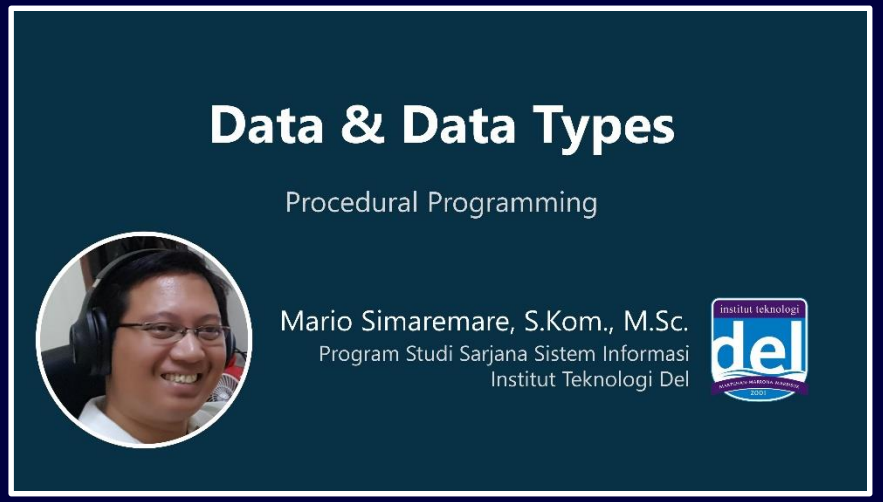
Objectives

- The objectives of this session are the following:
  - The students are able to elaborate the role of data types in a solution.
  - The students are able to choose the most appropriate data type to characterize value.
  - The students are able to practice the concept of type casting, value auto-boxing and auto-unboxing.



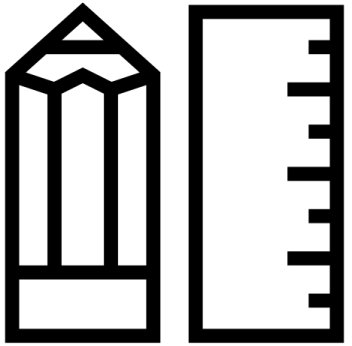
Please see these materials first





Outlines

1. Role of data and data type in a solution.
2. Data types.
3. Primitive data types.
4. Type conversion.
5. Wrapper classes for primitive types.
6. Auto-boxing and auto-unboxing.



The Role of Data & Data Type in a Solution

Data & data type in a solution

- Solution = data + algorithm.
  - Data are values that are meaningful in a specific context.
  - A value has a type that describe or characterize it accurately.

- A value can be used in various compatible operations.
  - The compatibility of an operation is dictates by the value type.
  - An operation might be fit for a type but not for the others.
  - Hence, it is very important to **choose** the best type for a value.

Data Types

Data types

- Java is a strongly-typed language.
  - Meaning every value must have a type that characterize the value.

- Java support a handful of types:
  - Primitive types.
    - Integer types, floating-point types, Boolean, and character.
  - Object-reference type.
    - It may have a set of properties and methods.
    - Based on class, interface, enumeration, etc.

Primitive Data Types

Integer types

- These types are used to characterize whole-numbers.
  - No unsigned version.
  - What if you need to store a value beyond the `long` type?

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Floating-point types

- These types are used to characterize floating-point numbers.
  - The `double` type provides a higher precision value.

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Boolean type

- This type is used to characterize Boolean value.
  - There are two possible values `true` and `false`.

Character type

- This type is used to characterize character value.
  - It is a 16-bit length value, compatible to `short` type.
    - Can be operated with number operators.
    - Support Unicode character set.

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal Unicode character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

Type Conversion

Type conversion

- It is possible to convert a value of a particular type into another value of another compatible type.
  - The conversion should preserve the actual information.

- Widening vs. narrowing conversion.
  - Widening conversion is storing a value to a type with larger spectrum, hence the original information is well preserved.
  - Narrowing is the opposite to widening conversion.
    This approach might cause an information lost.

Type conversion

- The rules for widening conversion are:
  - `byte` → `short` → `int` → `long` → `float` → `double`.

- Keep in mind that information lost would not cause any runtime exception.
  - It is the programmer responsibility to make sure the correctness of the program's behavior.


```
short s = 100;
float f = s + 1.02; // widening
int i = (int) f; // narrowing, information may lost here
```

Wrapper Classes for Primitive Data Types

Wrapper classes

- Wrapper classes are classes used to create wrapper objects for primitive values.
  - A primitive value **is not** an object.
  - Some classes require object references instead of primitives.
  - E.g. `java.utils.*`

- Every primitive type has its wrapper.
  - `int` → `Integer`, `char` → `Character`, etc.

- Wrapper objects are **immutable**.

Auto-{boxing|unboxing}

Auto-{boxing|unboxing}

- The terms refer to an automatic type conversion between primitive types to their wrappers and vice-versa.
  - Auto-boxing: converts primitives → objects.
    - An `int` value into an `Integer` object.
    - A `float` value into a `Float` object.
    - etc.
  - Auto-unboxing: converting objects → primitives.
    - An `Integer` object into an `int` value.
    - A `Character` object into a `char` value.
    - etc.

Auto-boxing an `int` into an `Integer` object

```
package example.datatype;

public class TypeExample {

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

        int pi = 10_000;
        builder.append(pi);
        boolean b = false;
        builder.append(b);
        char c = '\t';
        builder.append(c);
        float f = 31.18f;
        builder.append(f);
        Integer oi = pi;
        builder.append(oi);

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

}
```

Declaring and instantiating an object of `StringBuilder` class

Declaring and initializing a primitive (`boolean`) variable.

Auto-boxing an `int` value into an `Integer` object.

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 12 & JDK 12

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Modifier and Type	Method	Description
StringBuilder	append(boolean b)	Appends the string representation of the boolean argument to the sequence.
StringBuilder	append(char c)	Appends the string representation of the char argument to this sequence.
StringBuilder	append(char[] str)	Appends the string representation of the char array argument to this sequence.
StringBuilder	append(char[] str, int offset, int len)	Appends the string representation of a subarray of the char array argument to this sequence.
StringBuilder	append(double d)	Appends the string representation of the double argument to this sequence.
StringBuilder	append(float f)	Appends the string representation of the float argument to this sequence.
StringBuilder	append(int i)	nce.
StringBuilder	append(long lng)	ence.
StringBuilder	append(CharSequence s)	
StringBuilder	append(CharSequence s, int start, int end)	Appends a subsequence of the specified CharSequence to this sequence.
StringBuilder	append(Object obj)	Appends the string representation of the Object argument.
StringBuilder	append(String str)	Appends the specified string to this character sequence.
StringBuilder	append(StringBuffer sb)	Appends the specified StringBuffer to this sequence.
StringBuilder	appendCodePoint(int codePoint)	Appends the string representation of the codePoint argument to this sequence.
int	capacity()	Returns the current capacity.
char	charAt(int index)	Returns the char value in this sequence at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.

StringBuilder also accepts an instance of Object class. Integer itself is a child of the Object class

Another example of auto-{boxing|unboxing}

```
package example.datatype;

import java.util.ArrayList;
import java.util.List;

public class BoxingExample {

    public static void main(String[] args) {
        int pi = 10;

        List<Integer> list = new ArrayList<Integer>(2);

        list.add(pi);
        list.add(pi * 2);
        list.add(pi + 2);

        int pi3 = list.get(2);
        System.out.println(pi3);
    }

}
```

A primitive value, `10`.

Instantiating an object of `ArrayList` that is capable of storing `Integer` objects.

Storing some `int` values into the list. Auto-boxing happens here.

Retrieving value from the list at a particular index and store the value into an int variable. Auto-unboxing happens here.

This interface is a member of the Java Collections Framework.

Since:

1.2

See Also:

`Collection`, `Set`, `ArrayList`, `LinkedList`, `Vector`, `Arrays.asList(Object[])`, `Collections.nCopies(int, Object)`, `Collections.EMPTY_LIST`, `AbstractList`, `AbstractSequentialList`

Method Summary

All Methods | **Static Methods** | **Instance M**

The add method in the `List` interface accepts a specific type, in our case we specify it as `Integer` using the generic approach, the diamond symbols <>.

Modifier and Type	Method	
void	`add(int index, E element)`	Inserts the specified element at the specified position in this list (optional operation).
boolean	`add(E e)`	Appends the specified element to the end of this list (optional operation).
boolean	`addAll(int index, Collection<? extends E> c)`	Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
boolean	`addAll(Collection<? extends E> c)`	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
void	`clear()`	Removes all of the elements from this list (optional operation).
boolean	`contains(Object o)`	Returns `true` if this list contains the specified element.
boolean	`containsAll(Collection<?> c)`	Returns `true` if this list contains all of the elements of the specified collection.
static <E> List<E>	`copyOf(Collection<? extends E> coll)`	Returns an unmodifiable List containing the elements of the given Collection, in its iteration order.
boolean	`equals(Object o)`	Compares the specified object with this list for equality.
E	`get(int index)`	Returns the element at the specified position in this list.
int	`hashCode()`	

References

- Cay Horstman. Core Java.
- Matt Weisfeld. The Object-Oriented Thought Process.
- Java Language Specification, Ch. 5: Conversion and Contexts
  - https://docs.oracle.com/javase/specs/jls/se12/html/jls-5.html

Thank you