FACULTY OF ENGINEERING
Department of Telecommunications and Electronics Engineering

WRAPPER CLASS

Basic Java Course

Advisor: Prof. Thap Thareoun

Present By : Ngouch HongCheng
: Saing LymChhun

DEFINITION

- **A wrapper class** in Java is a class that wraps a primitive data type into an object. Since Java is an object-oriented programming language, it treats everything as objects, but primitive types (like int, char, double, etc.) are not objects.
- **Wrapper classes** allow primitive data types to be used as objects, enabling them to work in contexts where objects are required, such as in collections, generics, or reflection.

LIST OF WRAPPER CLASSES

- Each primitive type (e.g., int, char, boolean) has a corresponding wrapper class (e.g., Integer, Character, Boolean) in Java.
- These wrapper classes allow primitive types to be used as objects, enabling functionality such as **autoboxing**, **unboxing**, and the ability to store null values, which primitives cannot do.

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

1. DECLARING WRAPPER OBJECTS

- **Wrapper classes** are used to represent primitive types as objects. For example, Integer, Double, and Character are wrapper classes for int, double, and char respectively.
- **Wrapper classes** allow primitives to be used where objects are required, such as in collections or as parameters in methods.

Example

```
Integer obj = 10;  // Autoboxing primitive int to Integer
Double obj = 3.14; // Wrapper class for double
```

2. AUTOBOXING

- **Autoboxing** is the automatic conversion of a primitive type to its corresponding wrapper class. This happens when you assign a primitive to a wrapper object or use it in a collection that expects an object.
- This makes it easier to work with collections that require objects but need to store primitive data.

Example

```
Integer obj = 5;  // int 5 is automatically boxed to Integer
```

3. UNBOXING

- **Unboxing** is the reverse process where a wrapper class object is converted to its corresponding primitive type. This occurs automatically when the wrapper is used in an arithmetic operation or assignment to a primitive variable.
- Java handles unboxing automatically, so you don't have to manually convert from wrapper classes to primitives.

Example

```
int primitiveValue = obj;  // Integer obj is automatically unboxed to int
```

4. USING VALUEOF() TO CONVERT STRING TO WRAPPER OBJECT

- The **valueOf()** method converts a string representation of a number into the corresponding wrapper class object. It's commonly used to convert user input or data from a file into an object.
- **ValueOf()** is the preferred method over constructors for converting strings into wrapper objects.

Example

```
Integer obj = Integer.valueOf("123");  // Converts String to Integer object
```

5. USING PARSEX() METHODS TO CONVERT STRING TO PRIMITIVE

- The **parseX()** methods (like parseInt() and parseDouble()) are used to convert a string directly into a primitive type, without the need for a wrapper object.
- **parseX()** is faster than valueOf() because it returns a primitive type directly.

Example

```
int num = Integer.parseInt("123");  // String to primitive int
```

6. GETTING MINIMUM AND MAXIMUM VALUES

- **Wrapper classes** provide constants MAX_VALUE and MIN_VALUE that represent the maximum and minimum values of the respective primitive types.
- These are useful when you need to know the bounds of the data type.
- It is useful for validation and boundary checking in calculations.

Example

```
System.out.println(Integer.MAX_VALUE);  // Outputs 2147483647
```

7. USING TOSTRING() FOR STRING REPRESENTATION

- The **toString()** method returns a string representation of the wrapper object.
- It's commonly used for logging or displaying values in a readable format.
- This method is inherited from the Object class and can be used for any object, including wrapper classes.

Example

```
String str = obj.toString();  // Integer to String
```

8. USING COMPARETO() FOR COMPARING WRAPPER OBJECTS

The **compareTo()** method compares two wrapper objects. It returns:

- 0 if the objects are equal,
- A negative value if the first object is less than the second,
- A positive value if the first object is greater than the second.
- ==> It is useful for sorting and comparisons.

Example

```
int result = a.compareTo(b);  // Returns -1 if a < b
```

END