WEEK 4

Predefined Classes

Alawiyah Abd Wahab

Outline

- Class library and packages.
- import statement
- Creating packages.
- Predefined classes in the standard class library
- Using Java Predefined classes

Learning Objectives

- To describe how class library & package is organised into packages & the differences between classes & packages.
- To use import statement
- To define and create objects from predefined classes in the standard class library
- To apply methods call from the created objects to perform certain tasks.

Class Libraries

- A *class library* - a collection of predefined classes that can be used to develop programs.
- Java contains an extensive library of predefined classes provided by the Java Development Kit (JDK) and is known as the "Application Programming Interface“ (API).
- These classes are divided into groups called *packages.*
- Package: defines several related classes, interfaces, exception, and errors.

Class Libraries

- All the Java API packages are prefixed with java or javax - See more at: *https://docs.oracle.com/javase/8/docs/api/* contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform.

Packages

- A package - a namespace that organizes a set of related classes and interfaces (conceptually similar to different folders on your computer).
- The packages represent the tasks most commonly associated with general-purpose programming.

Packages

- Packages can be split into sub-packages.
  - E.g.: the `java.util` package has a sub-package called `java.util.Scanner`.
  - A class in a sub-package has no more access to a class in the parent package (or vice versa) than it would to a class in a completely different package.

Packages

- Some of the important packages in the standard class library are:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
java.sql	Database access in JDBC
javax.xml.parsers	XML document processing

`import` Declarations

- It is sometimes called `import` "statements".
- When you want to use a class from a package, you could use its *fully qualified name*. Syntax:

```
import <name of the package>;
```

`import` Declarations

- E.g.: The package `java.util` contains many useful classes, such as `Scanner`. If you need to use it, you can write a declaration such as:

```
import java.util.Scanner;
```

- This `import` statement makes the definition of the `Scanner` class available to the compiler
- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

Creating a Package

- Steps involved:
  i. choose a name for the package. Common practice: use lowercased names of packages to avoid any conflicts with the names of classes, interfaces. E.g. “animals”.
  ii. put a **package s**tatement with that name at the top (the first line) of every source file.

Creating a Package

- Example:

```
/* File name : Animal.java */
package animals;
interface Animal {
    public void eat();
    public void travel();
}
```

Java Predefined Classes

- Following are among the most commonly used Java pre-defined classes that will be discussed in this chapter:
  - String
  - Math
  - Random
  - Scanner
  - Primitive type wrapper

Using Predefined Classes

- Steps involved:
  i. Declaring an object reference variable of type a predefined class.
  ii. Creating object of the predefined class.
  iii. Invoking some methods of the predefined class.

Object Creation

- Step 1: Declaration of object reference variable
  - Syntax:

```
<class name> <object reference variable>;
```

  - Example: `String quote;`
- Step 2: Object instantiation
  - Syntax:

```
<object reference variable> = new <class name> (arguments);
```

  - Example: `quote = new String("Work hard");`

Object Creation

- Step 1 and Step 2 in one statement
  - Syntax:

```
<class name> <object reference variable> =
new <class name>(arguments);
```

  - Example:

```
String quote = new String("Work hard");
```

Revision…

The `String` class

- The `String` class represents character strings.
- All string literals in Java programs, such as `"Ali"`, are implemented as instances of this class.
- Thus, strings are actually `String` objects and constant; their values cannot be changed after they are created.
- Since the `String` class is a member of the *`java.lang`* package, NO import statement is necessary.

The `String` class

- String Methods: Commonly used Constructor

Constructor and Description
String() Initializes a newly created String object so that it represents an empty character sequence.
String(char[] value) Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

The `String` class: Object creation

- Declare and create object in two statements:

```
String name;
name = new String ("Najihah Najmy");
```

- Declare and create object in one statement:

```
String name = new String ("Najihah Najmy");
```

- `Strings` have a shortcut way to create an object

```
String name = "Najihah Najmy";
```

The `String` class

- Commonly used String methods

Return Type	Method and Description
char	charAt(int index)Returns the char value at the specified index.
int	compareTo(String anotherString)Compares two strings lexicographically.
int	compareToIgnoreCase(String str)Compares two strings lexicographically, ignoring case differences.
String	concat(String str)Concatenates the specified string to the end of this string.
boolean	equals(Object anObject)Compares this string to the specified object.
boolean	equalsIgnoreCase(String anotherString)Compares this String to another String, ignoring case considerations.
int	indexOf(int ch)Returns the index within this string of the first occurrence of the specified character.
int	indexOf(int ch, int fromIndex)Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	indexOf(String str)Returns the index within this string of the first occurrence of the specified substring.
int	indexOf(String str, int fromIndex)Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

The `String` class

- Commonly used String methods

Modifier and Type	Method and Description
String	toString()This object (which is already a string!) is itself returned.
String	toUpperCase()Converts all of the characters in this String to upper case using the rules of the default locale.
String	**valueOf**(**int i**) Converts an int into its string representation. The method is overloaded for all the primitive types.

Invoking Methods

- Once object is created, we can use the dot operator (.) to accessed its data or invoke its methods.
- Syntax:

```
<Object name>.<method name>(argument);
```

- Example:

```
System.out.println(quote.length());
```

Revision...

Invoking Methods: `String` class

```
public boolean equals(Object anObject)
```

BEHAVIOR: this method returns true if the String are equal; false otherwise.

- Example:

```
String word1 = "Welcome";
String word2 = "Welcome";
if(word1.equals(word2))
  System.out.println("They are equal");
else
 System.out.println("They are not equal");
```

- Output:

```
They are equal
```

Invoking Methods: `String` class

```
public int indexOf(char ch)
```

BEHAVIOR: this method returns the index within this string of the first occurrence of the specified character.

- Example:

```
String courseName = "Introduction to OOP";
int index = courseName.indexOf('o');
System.out.print("The index of 'O' is " + index);
```

- Output:

```
The index of 'O' is 4
```

The `Math` class

- The `Math` class contains mathematical methods that provide much of the functionality of a scientific calculator (E.g.: basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions).
- All the methods in the `Math` class are class methods (`static`), so they can be invoked without the need to create a `Math` object.
- These methods require argument (data) on which to perform their actions, and return a result that can be used in an expression in your program.

The `Math` class

- Two public class constants are available:
  - `PI` — the double value approximating π
  - `E` — the double value approximating e, the base for natural logarithms.
- Since the `Math` class is a member of the *`java.lang`* package, **NO** import statement is necessary.

The Math class

- Commonly used `Math` methods

Modifier and Type	Method and Description
static double	**abs**(double a)Returns the absolute value of a double value.
static float	**abs**(float a)Returns the absolute value of a float value.
static int	**abs**(int a)Returns the absolute value of an int value.
static long	**abs**(long a)Returns the absolute value of a long value.
static double	**cos**(double a)Returns the trigonometric cosine of an angle.
static double	**exp**(double a)Returns Euler's number *e* raised to the power of a double value.

The Math class

- Commonly used `Math` methods

Modifier and Type	Method and Description
static double	**floor**(double a)Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
static double	**log**(double a)Returns the natural logarithm (base *e*) of a double value.
static double	**max**(double a, double b)Returns the greater of two double values.
static float	**max**(float a, float b)Returns the greater of two float values.
static int	**max**(int a, int b)Returns the greater of two int values.

The Math class

- Commonly used `Math` methods

Modifier and Type	Method and Description
static long	**max**(long a, long b)Returns the greater of two long values.
static double	**min**(double a, double b)Returns the smaller of two double values.
static float	**min**(float a, float b)Returns the smaller of two float values.
static int	**min**(int a, int b)Returns the smaller of two int values.
static long	**min**(long a, long b)Returns the smaller of two long values.
static double	**pow**(double a, double b)Returns the value of the first argument raised to the power of the second argument.

The Math class

- Commonly used `Math` methods

Modifier and Type	Method and Description
static double	**sin**(double a)Returns the trigonometric sine of an angle.
static double	**sqrt**(double a)Returns the correctly rounded positive square root of a double value.
static double	**tan**(double a)Returns the trigonometric tangent of an angle.

The Math class

- Invoking methods
- Example from NetBeans :

The Random class

- The `Random` class is part of the `java.util` package
- Instances of the `java.util.Random` class are used to generate random numbers.
- It provides methods that generate pseudorandom numbers.
- All algorithmic random number generators actually produce pseudorandom numbers, not true random numbers. A pseudorandom number generator has a particular period, based on the nature of the algorithm used.

The Random class

- `Random` Method: Commonly used Constructor

Constructor and Description
Random() - Creates a new random number generator.

The Random class

- Commonly used `Random` Method

Modifier and Type	Method and Description
protected int	**next**(int bits)Generates the next pseudorandom number.
boolean	**nextBoolean**()Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
void	**nextBytes**(byte[] bytes)Generates random bytes and places them into a user-supplied byte array.
double	**nextDouble**()Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
float	**nextFloat**()Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

The Random class

- Commonly used `Random` Method

Modifier and Type	Method and Description
int	**nextInt**()Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
int	**nextInt**(int n)Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
long	**nextLong**()Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.

Invoking Methods: Random class

```
public int nextInt()
```

BEHAVIOR: this method returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence.

- Example:

```
Random randomNum = new random();
System.out.print("Random Number:");
System.out.println(randomNum.nextInt());
```

- Output:

```
Random Number:745199890
```

Invoking Methods: Random class

```
public int nextInt(int n)
```

BEHAVIOR: this method returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

- Example:

```
Random myRandom = new Random();
System.out.print("Random Number:");
System.out.println(myRandom.nextInt(10));
```

- Output:

```
Random Number:8
```

The `Scanner` class

- The `Scanner` class is part of the `java.util` package
- An instance of `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.

The Scanner class

- `Scanner` Method: Commonly used Constructor

Constructor and Description
Scanner(**File** source)Constructs a new Scanner that produces values scanned from the specified file.
Scanner(**String** source)Constructs a new Scanner that produces values scanned from the specified string.

The Scanner class

Modifier and Type	Method and Description
boolean	hasNext()Returns true if this scanner has another token in its input.
String	next()Finds and returns the next complete token from this scanner.
byte	nextByte()Scans the next token of the input as a byte.
double	nextDouble()Scans the next token of the input as a double.
float	nextFloat()Scans the next token of the input as a float.
int	nextInt()Scans the next token of the input as an int.
String	nextLine()Advances this scanner past the current line and returns the input that was skipped.
long	nextLong()Scans the next token of the input as a long.
short	nextShort()Scans the next token of the input as a short.

Invoking Methods: Scanner class

```
public String next()
```

BEHAVIOR: Finds and returns the next complete token from this scanner.

- Example:

```
Scanner input = new Scanner(System.in);
System.out.print("Please enter your name: ");
String name = input.next();
```

- Output:

```
Please enter your name: Nurul
```

Invoking Methods: Scanner class

```
public double nextDouble()
```

BEHAVIOR: this method scans the next token of the input as a `double`.

- Example:

```
Scanner input = new Scanner(System.in);
System.out.print("Please enter your score:");
double score = input.nextDouble();
```

- Output:

```
Please enter your score:89.9
```

The `Wrapper` class

- Java gives 8 primitive data type (double float long int short byte char boolean), i.e. given by the compiler.
- In event driven programmer there are many methods that require an object not a primitive.
- Thus, Java wraps these to give you an object of a primitive type.

The `Wrapper` class

- Wrapper classes "wraps" the value of a primitive data type into an object.
- Useful when methods require an object argument.
- Also useful for converting Strings to an `int` or `double`.

The Wrapper class

Mapping between primitive data and wrapper classes in Java API:

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

The Wrapper class

Below is Java API wrapper class hierarchy :

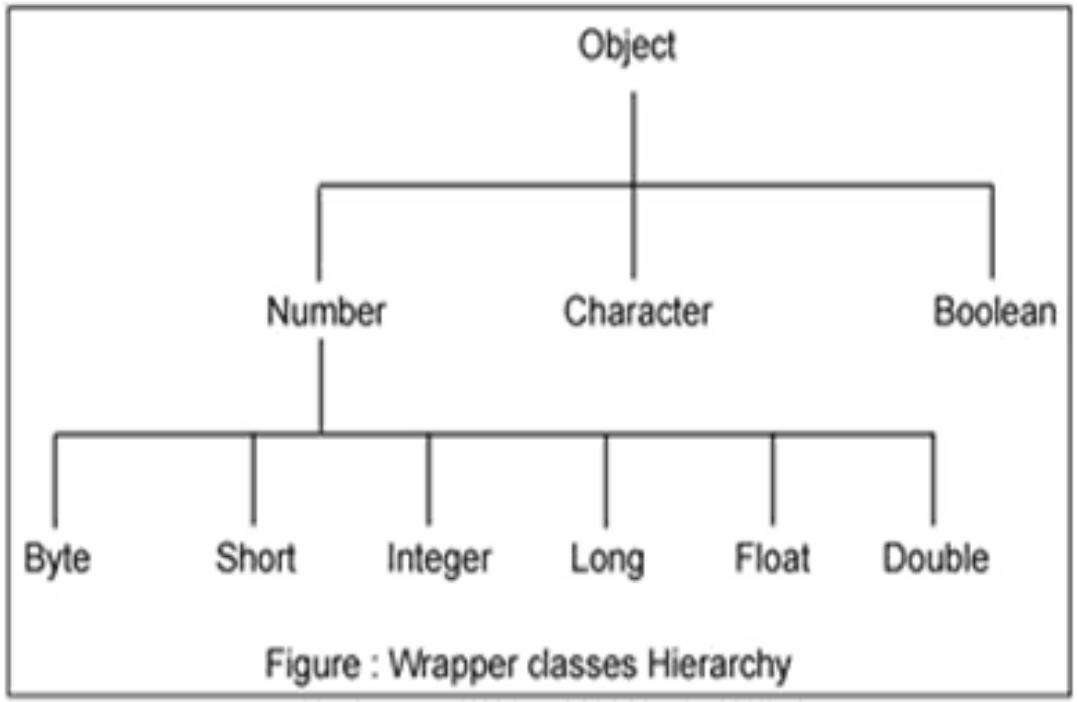

Figure : Wrapper classes Hierarchy

The Wrapper class

Return value	Method Name and argument list
`int`	`parseInt( String s )` returns the *String s* as an *int*
`Integer`	`valueOf( String s )` returns the *String s* as an *Integer* object

Return value	Method Name and argument list
`double`	`parseDouble( String s )` returns the *String s* as a *double*
`Double`	`valueOf( String s )` returns the *String s* as a *Double* object

Summary

- Predefined classes are the classes provided by a class library that are organised in packages.
- In Java platform, these classes are available via JDK and it is called Java API.
- Before any Java programmers can use this classes, they need to include a particular import statements and also create their own packages as a part of statements on the top lines of a source file.

Summary

- They can also invoke certain methods of a particular predefined class by creating an object of the class (for non-static methods) or using the class name representing the object name (for static method).