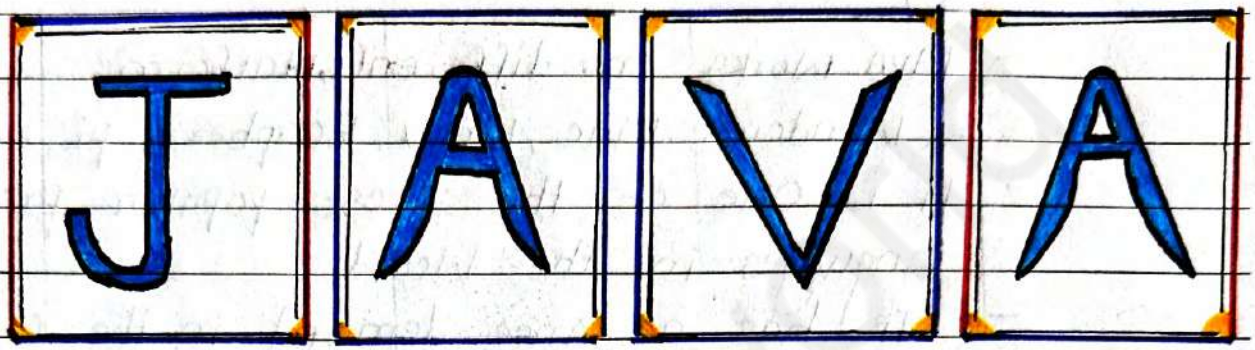
PAGE:
DATE: / /



JAVA INTRODUCTION :

What is JAVA?
JAVA is a popular programming language, created in 1995.

It is used for :

- Mobile applications (specially Android apps)
- Desktop applications.
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

Why Use JAVA?

- JAVA works on different platforms (Windows, Mac, Linux, Raspberry pi, etc.)
- It is one of the most popular programming languages in the World.
- It has a large demand in the current Job Market.
- It is easy to learn and simple to use
- It is open-source and free.
- It is secure, fast and powerful.
- JAVA is an object Oriented language which gives a clear structure to programs and allows code to be reused, lowering developement costs.
- As JAVA is close C++ and C#, it makes it easy for programmers to switch to JAVA or vice versa.

JAVA Syntax :

In previous chapter, we created a JAVA file called Main.java, and we used the following code to print "Hello World" to the screen:

Main.java

```
public class Main {
    public static void main (String [] args) {
        System.out.println( "Hello World" );
    }
}
```

JAVA Comments :

Comments can be used to explain JAva code, and to make it more readable. It can also be used to prevent execution When testing alternative code.

Single-line Comments

(Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed)

Example:

```
// This is a comment
System.out.println ("Hello World");
```

JAVA Variables :

Variables are containers for storing data values.

In JAVA, there are different types of variable for example:

String - Stores text, such as "Hello". String values are surrounded by double quotes.
Int - Stores integers (whole numbers), without decimals, such as 123 or 123
Float - Stores floating point numbers, with decimals, such as 19.99 or -19.99
Char - Stores single characteres, such as 'a' or 'B'. Char values are surrounded by single quotes
boolean - Stores value with two states: True or False.

Declaring Variables

Syntax:

```
type variableName = value;
```

Example:

Create a variable called name of type String and assign it the value "John":

```
String name = "John";
System.out.println(name);
```

JAVA Print Variables:

The println() method is often used to display variables.

To combine both text and a variable, use the + character.

Example:

```
String name = "John";
System.out.println("Hello" + name);
```

JAVA DATA Types:

A variable in Java must be a specified data type.

Example ↓

```
int myNum = 5;          // Integer (Whole number)
float myFloatNum = 5.99f; // Floating point number
char myLetter = 'D';    // Character
boolean myBool = true;  // Boolean
String myText = "Hello"; // String
```

JAVA Type Casting :

Type casting is when you assign a value of one primitive data typed to another type.

- Widening Casting (automatically) -
  Converting a Smaller type to a larger type size

byte -> Short -> char -> int ->
long -> Float -> double

Widening Casting

Widening casting is done automatically when passing a Smaller size type to a longer size type.

Example ↓

```
public class Main {
    public static void main (String[] args) {
        int myInt = 9 ;
        double myDouble  myInt ; // Automatic
                          casting : int to double

        System.out.println (myInt);  //Outputs 9
        System.out.println (myDouble); // Outputs 9.0
    }
}
```

JAVA Operators :

Operators are used to perform operations on variables and values.

In the example below, we use the + Operator to add together two values.

Example ↓

```
int x = 100 + 50;
```

JAVA Strings :

Strings are use for Storing text.

A String variable contains a collection of characters surrounded by quotes.

Example↓

Create a variable of String and assign it a value:

```
String greeting = "Hello";
```

String Length

A String in JAVA is actually an object, which contain methods that can perform certain operations on Strings. For example, the length of a String can be found with the length() Method.

Example↓

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the string is: " + txt.length());
```

JAVA Booleans :

Very often, in programming, you will need a data type that can only have one of two values, like;

- YES / NO
- ON / OFF
- TRUE / FALSE

For this JAVA has a boolean data type, which can store True or False values.

Boolean Values

A boolean type is declared with the boolean keyword and can only take the values True or False

Example ↴

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun); // Outputs true
System.out.println(isFishTasty); // Outputs false
```

JAVA IF...ELSE :

JAVA Conditions and If Statements

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to: a == b
- Not Equal to: a != b

Java has the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true.
- Use else to specify a block of code to be executed, if the same condition is false.
- Use else if to specify a new condition to test, if the first condition is false.
- Use switch to specify many alternative blocks of code to be executed.

The if Statement

Use the if Statement to specify a block of JAVA code to be executed if a condition

is True.

Syntax

```
if (condition) {
    // block of code to be executed if the
       condition is true
}
```

Example ↓

```
if (20 > 18) {
    System.out.println("20 is greater than 18");
}
```

JAVA Switch :

JAVA Switch Statement

Instead of writing many if... else statements, you can use the Switch statement.

The Switch statement selects one of many code blocks to be executed.

Syntax

```
Switch ( expression) {
  case x:
    // code block
    break;
  case y;
    // code block
    break;
  default :
    // code block
  }
```

This is how it Works:

- The switch expression is evaluated once.
- The value of expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional, and will be described later in this chapter.

Example ↓

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
} // outputs "Thursday" (day 4)
```

JAVA While Loop :

Loops

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

JAVA While Loop

The While loop loops through a block of code as long as a specified condition is True.

Syntax

```
While (condition) {
    // code block to be executed
}
```

Example ↓

```
int i = 0 ;
While (i < 5 ) {
```

```
        System.out.println(i);
        i++;
    }
```

JAVA FOR Loop :

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

Syntax

```
for (statement 1; statement 2; statement 3;) {
    // code block to be executed
}
```

Example ↓

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

JAVA Break And Continue :

JAVA Break

We already seen the break statement used

in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of loop.

Example↓

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

JAVA Arrays :

Arrays are use to store Multiple values in a single variable, instead of declaring seperate variables for each value.

To declare an array, define the variable type with square brackets.

```
String[] cars;
```

Example:

```
String [] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// outputs Volvo
```

JAVA METHODS :

JAVA Methods :

A Method is a block of code which only runs when it is called.

You can pass data, known as parameters, into a Method.

Methods are used to perform certain actions, and they are also known as functions.

Create a Method

A Method must be declared within a class. It is define with the name of the Method, followed by parentheses ().

Java provides some pre-defined Methods, such as System.out.println(), but you can also create your own Methods to perform certain actions.

Example:

```
public class Main {
    static void myMethod() {
        // code to be executed
    }
}
```

JAVA Method parameters:

Parameters And Arguements

Parameter act as variables inside the Method.

parameters are specified after the method name, inside the parenthese. We can add as many parameters as you want. Just separate them with comma.

String

called Frame as parameter. When the method is called, We pass along a first Name, which is used inside the method to print the full Name.

Example↓

```
public class Main {
    static void myMethod ( String fname ) {
        System.out.println( fname + " Refsnes " );
    }
    public static void main ( String [] args ) {
        myMethod ( "Liam" );
        myMethod ( "Jenny " );
        myMethod ( "Ania " );
    }
}
// Liam Refsnes
// Jenny Refsnes
// Ania Refsnes
```

JAVA Method Overloading :

Method Overloading

With Method Overloading, multiple methods can

have the same name with different parameters.

Example ↓

```
int myMethod (int x)
float myMethod (float x)
double myMethod (double x, double y)
```

JAVA SCOPE :

In Java, variables are only accessible inside the region they are created. This is called scope.

Example ↓

```
public class Main {
  public static void main (String[] args ) {

  // Code here CANNOT use x

    int x = 100;

  // Code here can use x
  System. out. println (x);
```

```
    }
}
```

JAVA Recursion :

Recursion is the technique of making a function call itself. This Teqnique provides a way to break complicated problems down into simple problems which are easier to solve.

Example ↓

```
public class Main {
    public static void main (String[] args) {
        int result = sum (10);
        System.out.println (result);
    }

    public static int sum(int k) {
        if ( k > 0 ) {
            return k + sum ( k - 1 );
        } else {
            return 0 ;
        }
    }
}
```

JAVA CLASSES :

JAVA OOP :

JAVA - What is OOP?
OOP stands for Object - Oriented programming.

Procedural programming is about writing procedures or methods that perform operations on the data, while Object- Oriented programming is about creating objects that contain both data and methods.

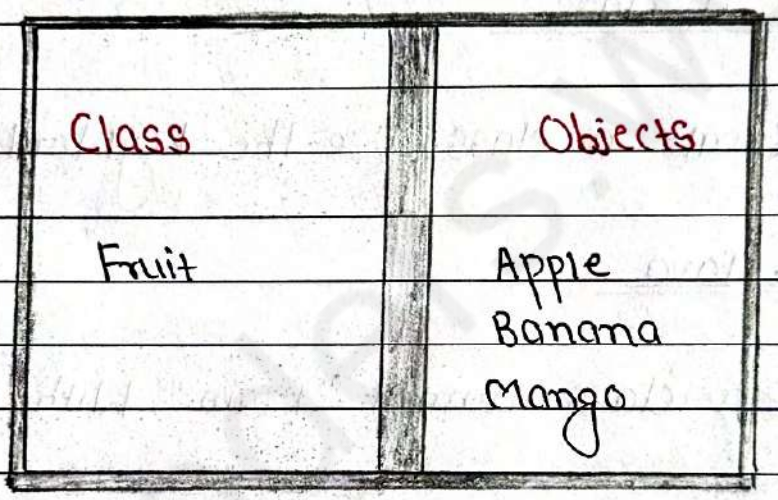
Object Oriented programming has several advantages over procedural programming.

- OOP is faster and easier to execute.
- OOP provides a clear structure for the programs.
- OOP helps to keep the Java code DRY "Don't Repeat yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possiple to create full reusable applications with less code and shorter development time.

JAVA - What are Classes and Objects?

Classes and Objects are the two main aspects of object-oriented programming.

Look at following illustration to see the difference class and objects.



JAVA Classes And Objects :

JAVA Classes / Objects

JAVA is an Object-Oriented programming language.

Everything in JAVA is associated with classes and objects, along with its attributes and methods. For example : in real life, a car is an object. The car has attributes, such as

Weight and color, and Methods, such as drive and brake.

A class is like an Object Constructor, or a "blueprint" for creating objects.

Create a class

To create a class, use the keyword class:

Main.Java

Create a class named "Main" with variable x:

```
public class Main {
    int x = 5 ;
}
```

Create An Object

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the

Class name, followed by the Object name, and use the keyword new:

Example↓

Create an object called "myObj" and print the value of x.

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Multiple Objects

Example↓

```
public class Main {
  int x = 5

  public static void main(String[] args {
    Main myObj1 = new Main(); // Object 1
```

```
Main myObi2 = new Main (); // Object 2
System.out.println (myObi1.x);
System.out.println (myObi2.x);
```

JAVA Class Attributes :

Example 1

Create a class called "Main" with two attributes x and y.

```
public class Main {
    int x = 5;
    int y = 3;
}
```

Accessing Attributes

Example 1

```
Public class Main {
    int x = 5 ;

public static void main (String[] args ) {
    Main myObi = new Main
```

```
        System.out.println(myObj.x);
    }
}
```

Modify Attributes

Example:

```
public class Main {
    int x;

    public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
    }
}
```

Multiple Attributes

```
public class Main {
    String fname = "John";
    String fname = "Doe";
    int age = 24;
```

```
public static void main (String[] args ) {
  Main myobj = new Main ();
  System.out.println ("Name: " + myobj.fname +
                      " " + myobj.lname );
  System.out.println ("Age: " + myobj.age);
  }
}
```

JAVA Class Methods :

Example ↓

```
Public class Main {
   Static void myMethod () {
      System.out.println ("Hello World!");
   }
}
```

Static v.s Public

JAVA programs have either Static or Public attributes and methods.

The example below, We created a Static

Method, which means that it can be accessed without creating an Object of the class, unlike public, which can only be accessed by Objects.

Example ↓

```
public class Main {
  // Static method
Static void myStaticMethod () {
  System.out.println ("Static methods can be
                       called without creating
                         Objects");
}

  // public Method
  public void myPublicMethod () {
  System.out.println ("public methods must be
                       called by creating Objects");
}

  // Main Method
public Static void main (String [] args) {
  myStaticMethod (); // Call the Static Method
  // myPublicMethod (); This would compile an error
```

```
Main myobi = new Main (); //Create an object main
myobi.mypublicMethod (); // Call the public
                            Method on the object
    }
}
```

JAVA Constructors :

A Constructor in Java is a special Method that is used to initialize objects. The Constructor is called when an object of a class is created. It can be used to set values for object attributes.

Example 1

```
// Create a Main class
public class Main {
  int x ; // Create a class attribute

// Create a class constructor for the Main
class
   public Main () {
      x =5; // Set the initial value for the
  Class attribute x
}
```

```
public static void main (String [] args {
    Main myobi = new main (); // Create an object
    class Main (This will call the (onstructor)
    System.out.println (myObi.x); // print the value
                                    of x
        }
}
```

// Outputs 5

Constructor parameters

Constructor can also take parameters, which is used to initialize attributes.

The following example adds an int y parameter to the constructor. Inside the constructor we set x to y (x=y). When we call the constructor,

Example ↓

```
public class Main {
  int x;
```

```
public Main(int y) {
    x = y;
}

public static void Main(String[] args) {
    Main myObj = new Main(5);
    System.out.println(myObj.x);
}
}

// Outputs 5
```

JAVA Modifiers :

Modifiers

```
public class Main
```

The public keyword is an access modifier, meaning that it is used to set the access level for classes, attributes, Methods and constructors.

- Access Modifiers - controls the access level

- Non-Access Modifiers - do not control access level but provides other functionality.

Access Modifiers

Modifier	Description
public	The class is accessible by any other class.
default	The class is only accesible by classes in the same package. This is used when you don't specify a modifier.

for attributes, methods and constructors, you can use the one of following.

Modifier	Description
public	This code is accessible for all classes.

private	The code is only accessible within the declared class.
default	The code is only accessible in the same package. This is used when you don't specify a modifier.
protected	The code is accessible in the same package and subclasses.

Non-Access Modifiers

Modifier	Description
final	The class cannot be inherited by other classes
Abstract	The class cannot

be used to create objects (To Access An abstract class, it must be inherited from another class.)

For attributes and Methods, you can use the one of the following.

Modifier	Description
Final	Attributes and Methods cannot be overridden / modified.
Static	Attributes and Methods belongs to the class, rather than an Object.
Abstract	Can only be used in an abstract class, and can only be used on Methods. The method does not have a body.

For example
abstract void run();
The body is provided by the subclass.

Final

Example 1

```
Public class Main {
    final int x = 10;
    final double PI = 3.14;

Public Static Void main (String [] args ) {
    Main myobi = new Main ()
    myobi . x = 50; // Will generate an error: Cannot
                       assign a value to final Variable
    myobi.PI = 25; // Will generate an error :
                       Cannot be assign a value
                       to a final variable.
    System.out.println (myobi.x);
    }
}
```

JAVA Encapsulation:

Encapsulation

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables / attributes as private
- provide public get and set methods to access and update the value of private variable.

Get And Set

private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set method.

The get method return the variable value, and the Set Method sets the value.

Syntax for both is that they start with either

get or set, followed by the name of the variable, with the first letter in Upper case.

Example

```
Public Class person {
   private String name; // private = restricted
                           access

   // Getter
   public String getName () {
      return name;
   }

   // Setter
   public void setName (String newName) {
      this.name = newName;
   }
}
```

JAVA Inheritance:

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- Subclass (child) - the class that inherits from another class
- Superclass (parent) - the class being inherited from

To inherit from a class, use the extends keyword.

In the example below, the car class (subclass) inherits the attributes and methods from the Vehicle class. (Superclass).

Example ↓

```
class Vehicle {
    protected String brand = "Ford";
                                // Vehicle Attributes
    Public void honk () {       // Vehicle Method
        System.Out.println("Tuut, tuut");
    }
}

Class Car extends Vehicle
    private String modelName = "Mustang"; // Car Attribute
    public static void main (String[] args {
```

```
// Create a my car object
Car myCar = new Car();

// Call the honk() method (from the vehicle
                (class) on the myCar object.
myCar.honk();

// Display the value of the brand Attribute
 (from the vehicle (class) and the
 modelName from the Car class.
System.out.println(myCar.brand + " " +
                        myCar.modelName);
    }
}
```

★ Why And When To Use "Inheritance"?

- It is useful for code reusability. reuse attributes and Methods of an existing class When you create a new class.

The Final Keyword

```
final class vehicle {
  ...
}
class Car extends vehicle {
```

```
}...
```

Output will be something like this ↓

```
Main.java:9: error: cannot
inherit from final vehicle
Class Main extends vehicle {
                   ^
1 error)
```

Java PolyMorphism :

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter, Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Example:

```
class Animal {
    public void animalSound () {
        System.out.println("The animal makes
                            a sound");
    }
}

class pig extends Animal {
    public void animalSound () {
        System.out.println("The pig says:
                            Wee Wee");
    }
}

class Dog extends Animal {
    public Void animalSound () {
        System.out.println("The dog says:
                            bow bow");
    }
}
```

* Why And When To Use "Inheritance" and "polymorphism"?

→ It is useful for code reusability: reuse

attributes and Methods of an existing class when you create a new class.

JAVA Abstraction :

Abstract Classes AND Methods

DATA Abstraction is the process of hiding certain details and showing Only essential information to the user.
Abstraction can be achieved With either abstract classes or interfaces.

The abstract keyword is an non-access Modifier, used for classed And Methods:

- Abstract class - is a restricted class that cannot be used to create Objects (to access it, it must be inherit from another class).

- Abstract Method - can only be used in an abstract class, and it does not have a body. The body is provided by the subclass. (inherit from)

Example 7

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
 public abstract void animalSound ();
  // Regular method
 public void Sleep () {
   System.out.println ("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
   public void animalSound () {
 // The body of animalSound () is provided here
  System.out.println("The pig says: Wee Wee");
  }
}

class Main {
  public static void main (String [] args) {
   Pig myPig = new Pig (); // Create a Pig object
   myPig.animalSound ();
   myPig.Sleep ();
  }
}
```

JAVA Interface !

Interfaces

Another Way to achieve abstraction in Java is With interfaces.

An interface is a completely "abstract class" that is used to group related methods With empty bodies.

Example ↓

```
// Interface
interface Animal {
  public void animalSound (); // Interface
                              Method (does not have body)
  Public void Sleep(); // interface Method
                         (does not have a body)
}

// pig "implements" the Animal interface
class pig implements Animal {
  Public void animalSound () {
// The body of animalSound () is provided here
System.out.println ("The pig says : Wlee Wlee");
```

```
}
public void sleep () {
  // The body of sleep () is provided here
  System.out.println ("Zzz");
}
}

Class Main {
  Public Static void main (String [] args ) {
  Pig mypig = new Pig (); // Create a pig
                             Object
  mypig.animalsound ();
  mypig.sleep ();
  }
}
```

JAVA Enums :

ENUMS

An enum is a special "class" that represents a group of constants (unchangeable variables like final variables).

To create an enum, use the enum keyword seperate the constants with a comma.

Example ↓

```
enum Level {
    LOW,
    MEDIUM,
    HIGH,
}
```

enum constants with dot syntax.

```
Level myVar = Level.MEDIUM;
```

Enum inside a Class

Example ↓

```
Public class Main {
  enum Level {
    LOW,
    MEDIUM,
    HIGH,
  }
```

```
public static void Main (String[] args) {
  Level MyVAr = Level. MEDEUM;
  System.out.println (myVar);
  }
}
```

JAVA User Input :

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, Create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings.

Example ↓

```
import java.util.Scanner; // Import the Scanner
                             class
class Main {
  public static void main (String[] args ) {
```

```
Scanner myObj = new Scanner (System.in);
                // Create a Scanner Object
System.out.println ("Enter username");

    String userName = myObj.nextLine ();
                // Read user input
System.out.println ("Username is: " + userName);
                // Output use input

    }
}
```

Input Types

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value

from the user

nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a Long value from the user
nextshort()	Reads a Short value from the user

Example ↴

```
import java.util.Scanner;

class Main {
    public static void Main (String[] args) {
      Scanner myobj = new Scanner (System.in);

      System.Out.println ("Enter name, age and
                           Salary:");

      // string input
```

```
String name = myobi.nextLine();

// Numerical input
int age = myobi.nextInt();
double Salary = myobi.nextDouble();

// Output input by user
System.out.println("Name: " + name);
System.out.println("Age: " + age);
System.out.println("salary: " + Salary);
    }
}
```

JAVA DATE AND TIME :

JAVA Dates

Java does not have a built-in Date class, but we can import the java.time package to work with the date and time API. The package includes many date and time classes.

ocean group
PAGE:
DATE: / /

Class	Description
LocalDate	Represents a date (year, month, day).
LocalTime	Represents a time (hour, minute, second. and nanoseconds)
LocalDateTime	Represents both a date and a time
DateTimeFormatter	Formatter for displaying and parsing date-time Objects.

Display current Date

To display the current date, import the java.time.LocalDate class, and use the now() method.

Example↓

```
import java.time.LocalDate; import the LocalDate class
```

```
public class Main {
    public static void main (String[] args) {
        LocalDate myobi = LocalDate.now ();
                    // create a date object
        System.out.println (myobi );
                    // Display the current date.
    }
}
```

The ofpattern() method accepts all sorts of values, if you want to display the date and time in a different format.

Example:

Value	Example
yyyy-MM-dd	"1988-09-29"
dd/MM/yyyy	"29/09/1988"
dd-MMM-yyy	"29-Sep-1988"
E, MMM dd yyyy	"Thu, Sep 29 1988"

JAVA Array List :

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified. While elements can be added and removed from an ArrayList whenever you want.

Example↓

Create an ArrayList object called cars that will store strings:

```
import java.util.ArrayList; // import the ArrayList class
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object.
```

Add Items

The ArrayList class has many useful methods. For example, to add elements to the

ArrayList. Use the Add() method.

Example ↓

```
import java.util.ArrayList;

Public class Main {
    public static void main (String[] args) {
        ArrayList<String> cars = new ArrayList<
                                    String>();
    cars.add("volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
    }
}
```

Access an Item

To access an element in the ArrayList, use the get() method and refer to the index number.

Example → cars.get(0);

Change An Item

To modify an element, use the set() method and refer to the index number.

Example → cars.set(0, "Opel");

Remove an Item

To remove an element, use the Remove() method and refer to the index number.

Example → cars.remove(0);

Arraylist size

To find out how many elements an ArrayList have, use the size method.

Example → cars.size();

Loop Through an Arraylist

Loop through the elements of an ArrayList

With a for loop, and use the size() method to specify how many times the loop should run.

Example ↓

```
public class Main {
  public static void main (String [] args ) {
    ArrayList <String> cars = new ArrayList
                              <String> ();
  cars. add ( "Volvo" );
  cars. add ( "BMW" );
  cars. add ( "Ford " );
  cars. add ("Mazda");
  for (int i = 0; i < (cars. size () i ) {
    System. out. println ((cars. get (i));
```

JAVA LinkedList :

You learned about the ArrayList class. The LinkedList class is almost identical to the Arraylist.

Example ↓

```
// Import the LinkedList class
import java.util.LinkedList;

public class Main {
  public static void main(String[] args) {
    LinkedList<String> cars = new LinkedList
                              <String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

ArrayList vs. LinkedList

The LinkedList class is a collection which can contain many objects of the same type, just like the ArrayList.

The LinkedList class has all of the same methods as an ArrayList class because they both implement the List interface.

This means that you can add items, change items, remove items and clear the list in the same way.

However, while the ArrayList class and the LinkedList class can be used in the same way, they are built very differently.

How the ArrayList Works

The Arraylist class has a regular array inside it. When an element is added, it is placed into the array. If the array is not big enough, a new, larger array is created to replace the old one and the old one is removed.

How the LinkedList Works

The LinkedList stores its items in "containers". The list has a link to the first container and each container has a link to the next container in the list. To add an element to the list, the element is placed into a new container and that container

is linked to one of the other containers in the list.

* When To Use

Use an ArrayList for storing and accessing data, and LinkedList to manipulate data.

LinkedList Methods

For many cases, the ArrayList is more efficient as it is common to need access to random items in the list, but the LinkList provides several methods to do certain operations more effecienlty.

Method	Description
addFirst()	Adds an item to the beginning of the List.
addLast()	Add an item to the end of the List.

removeFirst()	Remove an item from the beginning of the list
removeLast()	Remove an item from the end of the list
getFirst()	Get the item at the beginning of the list
getLast()	Get the item at the end of the list

JAVA HASHMAP :

In the ArrayList Chapter, we learned that Arrays Store items as an Ordered collection, and you have to access them with an index number (int type). A HashMap however, Store items in "key/value" pairs, and you can access them by an

index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types. String keys and Integer values, or the same type, like String keys and String values.

Example ↓

Create a HashMap object called capitalCities that will store String keys and String values.

```
import java.util.HashMap; // import the
                          HashMap class
HashMap<String, String> capitalCities = new
                          HashMap<String, String>
                          ();
```

Add Items

The HashMap class has many useful methods. For example, to add items to it, use the put() method.

Example ↓

```
// Import the HashMap class
import java.util.HashMap;

Public class Main {
    Public Static void main (String[] args) {
// Create a HashMap object called capitalCities
HashMap<String, String> capitalCities = new
                        HashMap<String, String>();

// Add keys and values (country, city)
capitalCities.put ("England", "London");
capitalCities.put ("Germany", "Berlin");
capitalCities.put ("Norway", "Oslo");
capitalCities.put ("USA", "Washington DC");
    System.Out.println (capitalCities);
    }
}
```

Access an Item

To access a value in the HashMap, use the get() method and refer to its key.

Example → capitalCities.get ("England");

Remove An Item

To remove an item, use the remove() Method and refer to the key.

Example → CapitalCities.remove("England");

HashMap Size

To find out how many items there are, use the Size() Method.

Example ↓

```
CapitalCities.size();
```

Loop Through a HashMap

Loop through a HashMap items of a HashMap with a for-each loop.

Example ↓

```
// Print keys
for (String i : capitalCities.keySet()) {
  System.out.println(i);
}
```

JAVA Wrapper Classes :

Wrapper classes provide a way to use primitive data types (int, boolean, etc) as objects.

The table below shows the primitive type and the equivalent Wrapper class.

primitive Data Type	Wrapper Class
byte	Byte
Short	Short
int	Integer
long	Long
Float	Float
double	Double
boolean	Boolean
Char	Character

Example ↓

```
ArrayList<int> myNumbers = new ArrayList<int>
                           (); // Invalid

ArrayList<Integer> myNumbers = new ArrayList
                           <Integer>(); // Valid
```

Creating Wrapper Objects

To create a Wrapper Object, use the Wrapper class instead of the primitive type. To get the value, you can just print the object.

Example ↓

```
public class Main {
  public static void main(String[] args ) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
```

}

JAVA Exceptions - Try .... catch :

JAVA Exceptions

When executing Java code, different errrows can occure: Coding errors mady by the programmer, error due to Introng input, or other Unforeseeable things.

When an error occurs, Java will normally stop and generate an error message - The technical term for this is : Java will throw an exception.

JAVA try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

Syntax ↓

```
try {
  // Block of code to try
}
catch (Exception e) {
  // Block of code to handle errors
}
```

Example ↓

```
public class Main {
  public static void main (String[] args) {
    try {
      int[] myNumbers = { 1, 2, 3 };
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

Finally

The finally statement lets you execute code, after try ... catch, regardless of the

result.

Example ↓

```
public class Main {
  public static void main(String[] args ){
    try {
      int[] myNumbers = { 1, 2, 3 };
System.out.println(myNumbers[10]);
    } catch (Exception e) {
System.out.println("Something went wrong");
} finally {
  System.out.println("The 'try catch' is
                      finished.");
    }
  }
}
```

Output → Something went wrong.
The 'try catch' is finished.

The Throw keyword

The Throw statement allows you to create a custom error.

The throw statement is used together with an exception type. There are many exception types available in java. ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.

Example ↓

Throw an exception if age is below 18 (print "Access denied"). If age is 18 or older, print "Access granted".

```
public class Main {
  static void checkAge( int age ) {
    if (age < 18 ) {
      throw new ArithmeticException ("Access
denied - you must be at least 18 year old.");
    }
    else {
      System.out.println ("Access granted - you are
                                old enough!");
    }
}

  public static void main (String [] args ) {
    CheckAge (15);
```

```
        // set age to 15 (which is below 18...)
    }
}
```

JAVA Threads :

Threads allows a program to operate more efficiently by doing multiple things at the same time.

Threads can be used to perform complicated tasks in the background without interrupting the main program.

Creating a Thread

There are two ways to create a thread.

It can be created by extending the Thread class and overriding its run() method.

Extend syntax →

```
public class Main extends Thread {
    public void run () {
        System.out.println("This code is running in thread );
    }
}
```

Another way to create a thread is to implement the Runnable interface.

Implement Syntax

```
public class Main implements Runnable {
  public void run () {
    System.out.println ("This code is running
                         in a thread");
  }
}
```

Running Threads

If the class exends the Thread class, the thread can be run by creating an instance of the class and call its start() method.

Extend Example ↓

```
public class Main extends Thread {
  public static void main (String [] args ) {
  Main thread = new Main ();
  thread.Start ();
  System.out.println ("This code is outside of
                       the thread");
```

```
}
public void run () {
    System.out.println ("This code is running
                         in a thread");
    }
}
```

JAVA File Handling :

JAVA Files :

File handling is an important part of any application.

Java has several methods for creating, reading, updating, and deleting files.

JAVA File Handling

The File class from the java.io package, allows us to work with files.

To use the File class, create an object of

the class, and specify the filename or directory name.

Example:

```
import java.io.File ; // Import the file class

File myobj = new File ("filename.txt");
                    // specify the Filename
```

The File class has many useful methods for creating and getting information about files. For example.

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not.

createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutepath()	String	Returns the absolute pathname of the file.
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the dictionary.

mkdir () Boolean (creates a directory

JAVA Create and Write To Files :

Create a File

To create a file in Java, you can use the createNewFile () method. This method returns a boolean value. true if the file was successfully created, and False if the file already exists. Note that the method is enclosed in a try .... catch block. This is necessary because it throws an IOException if an error occurs.

Example ↴

```
import java.io.File; // Import the File class
import java.io.IOException; // Import the
                    IOException class to handle error

public class Createfile {
  public static void main (String[] args ) {
  try {
```

```
File myobi = new File ("filename.txt");
if (myobi.createNewFile()) {
 System.out.println("File created: " + myobi.
                                    getName());
} else {
 System.out.println("file already exists.");
}

} catch (IOException e) {
 system.out.println("An error occurred.");
 e.printStackTrace();

   }
  }
}
```

Output → File created: filename.txt

Write to a file

Use the FileWriter class together with its write() method to write some text to the file we created in the example above.

Example ↓

```
import java.io.FileWriter; // Import the
                              FileWriter class
import java.io.IOException;
        // Import the IOException class
           to handle errors

public class WriteToFile {
    public static void main(String[] args) {
    try {

    FileWriter myWriter = new FileWriter
                ("filename.txt");

    myWriter.write("Files in java might be
                    tricky, but it is fun
                    enough!");
    myWriter.close();
    System.out.println("Successfully wrote to
                        the file.");
  } catch (IOException e) {
    System.out.println("An error occured.");
    e.printStackTrace();
    }
  }
}
```

Output → Successfully Wrote to the file.

Java Read Files :

Scanner class to read the the contents of the text file we created in the previous chapter.

Example ↴

```
import java.io.File; // Import the file class

Public class GetFileInfo {
   public Static void main (String[] args ) {
    File myobi = new File (" Filename.txt " );
 if (myobi.exists () ) {
    System.out.println (" file name : " + myobi.
                          getName ());
    System.out.println (" Absolute path : " +
                          myobi.getAbsolutepath () );
    System.out.println (" Writetable : " + myobi.
                          canWrite ());
    System.out println (" Readable " + myobi.
                          canRead ());
    System.out.println (" file size in bytes " +
```

```
        myObj.length());
    } else {
        System.out.println("The file does not
                            exist.");
        }
    }
}
```

Output → File name: filename.txt
Absolute path:
C:\Users\myName\filename.txt
Writeable: true
Readable: true
File size in bytes: 0

JAVA Delete Files :

Delete A File

To delete a file in java, use the delete() Method.

Example 1,

```
import java.io.File; // Import the file
                        class
```

```
public class Deletefile {
    public static void main (String [] args ) {
  File myobi. = new file (" filename. txt " );
  if ( myobi. delete ()) {
   System . out. println (" Deleted the file : "
                          + myobi. getName ());
  } else {
  System. out. println ( "failed to delete the file.");
          }
      }
  }
```

Output :-

Deleted the file: filename.txt