Java Reference

- This is a reference and a quick refresher on some of the main concepts of Java, and is not a substitute to the main course material.
- This is a live, not yet complete, document. (Version 1.0, 21 Sep 2022)
- More information on https://gituobdev.github.io/New_Developers/

Contents

Setup and First Program

- Java is a popular programming language. , created in 1995.
- It is owned by Oracle, and more than 3 billion devices run Java.
- Java is used to develop mobile apps, web apps, desktop apps, games … etc.
- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- Java code is first compiled into byte code (machine-independent code). Which runs on Java Virtual Machine (JVM) regardless of the underlying architecture.
- It is open-source and free

1-Open Command Prompt (cmd.exe)

```
C:\Users\Your Name>java -version
```

```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)
```

If not setup follow the following guide to set it up https://www.youtube.com/watch?v=cRgLuNWCq6c

2-Open any text editor (like Notepad), type the following and save it as Main.java, the same as the class name.

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

Note: Java is case-sensitive: "MyClass" and "myclass" is different.

3-compile your code

```
C:\Users\Your Name>javac Main.java
```

4-run

```
C:\Users\Your Name>java Main
```

- The name of the java file **must match** the class name.
- Every program must contain the **main()** method/function.
- The **curly braces {}** marks the beginning and the end of a block of code.
- **println()** method to print values in the command line
- Each code statement must end with a **semicolon (;)**

Comments

You can insert comment in the code that will not run. // for single line comments, and /**/ for multiline comments

```
// This is a comment
System.out.println("Hello World");
/* The code below will print the words Hello World
to the screen, and it is amazing */
System.out.println("Hello World");
```

Data Types and Variables

Variables are containers for storing data values.

Variables have a **name** "**myNum"** and a **type** "**int"**.
Giving a value to a variable is called **assignment**.

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

More variable types

```
int myNum = 5;               // Integer (whole number)
byte myNum = 100;
short myNum = 5000;
long myNum = 15000000000L;
float myFloatNum = 5.99f;    // Floating point number
double myNum = 19.99d;
char myLetter = 'D';         // Character
boolean myBool = true;       // Boolean
String myText = "Hello";     // String
```

You can define multiple variable in one line

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
int x, y, z;
x = y = z = 50;
System.out.println(x + y + z);
```

Java has 8 **primitive** (basic) data types like

Data Type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter or ASCII values

Note: Java also has **Non-primitive data** types - such as String, Arrays and Classes which we'll see later

floating point number can also be a scientific number with an "e" to indicate the power of 10

```
float f1 = 35e3f;
double d1 = 12E4d;
```

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only **six or seven** decimal digits, while double variables have a precision of about **15** digits.

Variable names rules:

- Names can contain letters, digits, underscores, and dollar signs
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like Java keywords, such as int or boolean) cannot be used as names

Type Casting

Type casting is when you assign a value of one primitive data type to another type.
Widening Casting (automatically) - converting a smaller type to a larger type size
byte -> short -> char -> int -> long -> float -> double

```
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double
```

Narrowing Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char -> short -> byte

```
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int
```

Operators (Arithmetic,Assignment,Comparison,Logical)

Operators are used to perform operations on variables and values.

Arithmetic operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

```
int x,y,z;
x = 14;
y = 5;

z = x / z; // 2
z = x % z; // 4
```

Assignment operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

```
int x,y;
boolean z;
x = 10;
y = 5;

z = x > 11 && y > 1; // false
z = x > 11 || y > 1; // true
```

Strings (Text)

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

Use the + operator to combine strings. This is called **concatenation.**

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

A String in Java is actually an object, which contain methods that can perform certain operations on strings

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

```
String txt = "Hello World";
System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"
System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

indexOf() method returns the index (the position) of the first occurrence of a specified text

0 is the first position in a string, 1 is the second, 2 is the third ...

More methods: https://www.w3schools.com/java/java_ref_string.asp

Strings and Numbers

```
int x = 10;
int y = 20;
int z = x + y;  // z will be 30 (an integer/number)
```

```java
String x = "10";
String y = "20";
String z = x + y;  // z will be 1020 (a String)
```

```java
String x = "10";
int y = 20;
String z = x + y;  // z will be 1020 (a String)
```

Escape Sequences

To include special charcater in a string, use the **backslash escape character**.

```java
String txt = "We are the so-called \"Vikings\" from the north.";
```

Escape character	Result
\'	'
\"	"
\\	\

Code	Result
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

Math

Math class has many methods that allows you to perform mathematical tasks on numbers.

```java
Math.max(5, 10);// 10
Math.min(5, 10);// 5
Math.sqrt(3);   // 9
Math.abs(-4.7); // 4.7
Math.random();  // returns a random number between 0.0 (inclusive), and 1.0 (exclusive)
```

```java
int randomNum = (int)(Math.random() * 101);  // 0 to 100
```

More methods: https://www.w3schools.com/java/java_ref_math.asp

Control Structures

Conditional Flow (If, Ternary Op.,Switch)

If: checks a condition and runs/executes code block if true

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Else if and else are optional.
Else is run if none of the conditions checks above it are true.

Ternary operator: is a shorthand for if{} else{} // the below outputs Good evening.

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

Switch: Similar to writing if{} else if{} else if{} .. else{} statements

```
int day = 4;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend"
```

default is like else.

break; is needed to break out of the switch block. Without it code below it will run regardsless if true or not;

Iterative Flow(While,For,ForEach)

While: Loop executes a block of code until condition is not true.

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

Do While: Like While but guaranteed to run at least once

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

For: Loop executes a block of code until condition is not true, like while.

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

int i = 0; is executed (one time) before the execution of the code block.
i < 5; defines the condition for executing the code block.
i++ is executed (every time) after the code block has been executed.

For Each: loops through elements in an array

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Break and Continue in For and While Loops

Break exits(jumps out of) the loop structure // output: 0,1,2,3

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

Continue skips code below and goes to the next iteration // output: 0,1,2,3,5,6,7,8,9

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

Arrays

Arrays are used to store multiple values in a single variable

```
int[] myNum = {10, 20, 30, 40};
```

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

```
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Array size

```
System.out.println(cars.length);
// Outputs 4
```

Loop through elements

```
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

```
for (String i : cars) {
  System.out.println(i);
}
```

Defining an empty array with a fied size

```
int intArray[];    //declaring array
intArray = new int[20];  // allocating memory to array
```

Multidimensional

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x); // Outputs 7
```

```
    for (int i = 0; i < myNumbers.length; ++i) {
      for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
```

Methods/Functions

A block of code which only runs when it is called.
Define the code once, and use it many times.

```
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Parameters/Arguments

Information that can be passed to methods.

```
public class Main {
  static void myMethod(String fname, int age) {
    System.out.println(fname + " is " + age);
  }

  public static void main(String[] args) {
    myMethod("Liam", 5);
    myMethod("Jenny", 8);
    myMethod("Anja", 31);
  }
}

// Liam is 5
// Jenny is 8
// Anja is 31
```

The method call must have the same number of arguments as there are parameters, in the same order.

Variable Scope

Variables are only accessible inside the region they are created (Method or Block{}) . This is called scope.

Return type

A method can return a value, we specify the returned value type before the method name, e.g. int

```
public class Main {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
// Outputs 8 (5 + 3)
```

If a method does not return anything we use void

```
  static void myMethod(String fname, int age) {
```

Overloading

Methods can have the same name as long as the number and/or type of parameters are different.

```
static int plusMethod(int x, int y) {
  return x + y;
}

static double plusMethod(double x, double y) {
  return x + y;
}
```

Recursion

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

```
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;
    }
  }
}
```

// Example used to add all of the numbers up to 10.

Be careful with recursion, it can be easy to slip into writing a function which never terminates.

Classes

So far, we did **Procedural programming** (writing procedures or methods). **Object-oriented programming (OOP)** is about creating objects from classes that contain both attributes and methods.

A **class** is a **template/blueprint** for objects, and an **object** is an **instance** of a class. Examples:

- Apple,Banana,Orange are objects of class Fruit
- Mazda, Toyota, Nissan are objects of class Car

Classes have **attirbutes** and **methods**.Example:

- Car has **attributes**, such as brand, weight and color, and **methods**, such as drive and brake.

1-Create the following class Fruit and save it as Fruit.java

```
public class Fruit {

    public String name;
    public boolean peeled = false;

    public void peel(){
        this.peeled = true;
        System.out.println("Peeled the " + " " + this.name);
    }

    public void eat(){
        if(this.peeled){System.out.println("Ate the " + this.name);}
        else{System.out.println("Please peel the " + this.name + " first");}
    }

}
```

2-Create the following class Main and save it as Main.java

```
public class Main {

  public static void main(String[] args) {

    Fruit obj1 = new Fruit();
    obj1.name = "Orange";
    obj1.eat();

    Fruit obj2 = new Fruit();
    obj2.name = "Apple";
    obj2.peel();
    obj2.eat();

  }

}
```

Note a class name should always start with an uppercase first letter, and that the name of the java file should match the class name.

3-Compile and run/execute Main

```
\desktop> javac Main.java Fruit.java
\desktop> java Main
```

Output

```
Please peel the Orange first
Peeled the  Apple
Ate the Apple
```

Constructor

A constructor is a special method that is used to initialize objects.
The constructor is automatically called when an object of a class is created.
The constructor name must match the class name, and it cannot have a return type
It can be used to set initial values for object attributes.
All classes have constructors by default even if you do not create one.

1-Add this inside the Fruit class

```
    // Fruit Constructor
    public Fruit(String name){
        this.name = name;
    }
```

2-Change Main.java as below

```
    Fruit obj1 = new Fruit("Orange");
    obj1.eat();

    Fruit obj2 = new Fruit("Apple");
    obj2.peel();
    obj2.eat();
```

```
Please peel the Orange first
Peeled the  Apple
Ate the Apple
```

Access Modifiers (public, private, protected, default)

So far, we have only used the public access modifier.
Access modifier set the access level for classes, attributes, methods and constructors

For classes, you can use the one of the following:

- **public** The class is accessible by any other class
- **default** If you don't specify anything, the class is only accessible by classes in the same package.

For attributes, methods and constructors, you can use the one of the following:

- **public** The code is accessible for all classes
- **private** The code is only accessible within the declared class

- **protected** The code is accessible in the same package and subclasses.
- **default** If you don't specify anything, the code is only accessible in the same package.

Encapsulation

Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, we must:

1. Declare class variables/attributes as private
2. Define public get and set methods to access and update the value of a private variable

```
public class Person {
  private String name; // private = restricted access

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

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

Note: With encapsulation, class attributes can be made **read-only** (if you only define the get method), or **write-only** (if you only define the set method)

Inheritance

Inheritence is used to define super classes (parents) and subclasses (children)
Example: Birds and Mammels are Subclasses of the Superclass Animal
Subclasses can use public and protected attrubutes and members of the super class
To inherit from a class, use the extends keyword. You can only extend from one class.

```java
class Animal{
protected String name = "This animal";
public void sleep(){System.out.println(this.name+ " sleeps");}
}

class Bird extends Animal { public void fly(){System.out.println(this.name + " flies");}}
class Mammel extends Animal{ public void run(){System.out.println(this.name + " runs");}}


public class Main {

  public static void main(String[] args) {

    Animal x = new Animal();
    x.sleep();

    Bird y = new Bird();
    y.sleep();
    y.fly();

    Mammel z = new Mammel();
    z.sleep();
    z.run();

  }

}
```

Remember name is protected as a family secret, it can only be accessed inside of Animal and Mammel
Function sleep() is public it can be acced inside of Animal, Mammel and Main
Inheritance is useful for code reusability

Polymorphism

Meaning "many forms", it allows subclasses to give different implementation to inherited methods.

```java
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 wow");
  }
}
```

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

Notice how we use widening type casting to assign objects Pig and Dog to variable of type Animal

Abstraction

Data abstraction is the process of hiding certain details and showing only essential information.

The **abstract** keyword is a non-access modifier, used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited 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 (inherited from).

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
Animal myObj = new Animal(); // will generate an error
// 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");
  }
}
```

Note: Abstraction can also be achieved with Interfaces

Inner Classess

It is possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.
Another advantage of inner classes, is that they can access attributes and methods of the outer class
Unlike a "regular" class, an inner class can be private or protected.

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

An inner class can also be static, so you can access it without creating an object of the outer class.

```
  static class InnerClass {
    int y = 5;
  }
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
```

Interfaces

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

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the **implements** keyword (**instead of extends**).

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

Like abstract classes, interfaces cannot be used to create objects
On implementation of an interface, you must override all of its methods
Interface methods are by default **public** and **abstract**
Interface attributes are by default **public**, **static** and **final**

Java does not support "multiple inheritance" (with extends) from more than one class. However, you can imlement multiple interfaces.

```
class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
```

Enums

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables). enum constants are public, static and final

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

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

The enum type has a values() method, which returns an array of all enum constants.

```
for (Level myVar : Level.values()) {
  System.out.println(myVar);
}
```

```
LOW
MEDIUM
HIGH
```

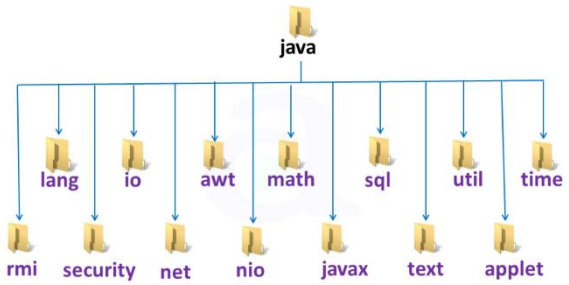
Packages & API

A package is used to group related classes. Packages are divided into two categories:

- Built-in Packages (packages from the Java API library)
- User-defined Packages (packages we create)

Built-in Packages

The Java API library contains components for managing input, database programming … etc.



Java API library package list:
https://docs.oracle.com/javase/8/docs/api/

To use a class in a package you can either import just the call from the package or the whole package.

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Example, we import the **Scanner** class which is used to get user input

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

The **java.util** package also contains date and time facilities, random-number generator and other utility classes.

User-defined Packages

To create a package, use the package keyword

1-Create files MyClass1.java and MyClass2.java as below

```
package mypackage;

public class MyClass1 { }
```

```
package mypackage;

public class MyClass2 { }
```

2-Then compile the package:

```
\desktop> javac -d . MyClass1.java
\desktop> javac -d . MyClass2.java
```

This forces the compiler to create the "mypackage" package folder.
The -d keyword specifies the destination. You can use any directory name, like c:/user (windows), or, use the dot sign ".",to compile in the current location.

Note: The package name should be written in lower case to avoid conflict with class names.
Note: Companies use their reversed Internet domain name to begin their package names—for example, com.example.mypackage for a package named mypackage created by a programmer at example.com.

Exceptions - Try...Catch

During execution, different errors can occur: coding errors, errors due to wrong input .. etc

When an execution error occurs, Java stops and **throws** an error ⚾🧢

```
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

To handle errors, we can use **try catch** and **finally**

```
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.");
    }
  }
}
```

```
Something went wrong.
The 'try catch' is finished.
```

We can also throw a **custom error**.

There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException .. etc.

```
if (password != "password") {
    throw new ArithmeticException("Access Denied!");
}
```

```
Exception in thread "main" java.lang.ArithmeticException: Access Denied!
        at Main.checkAge(Main.java:5)
        at Main.main(Main.java:14)
```

References

https://www.w3schools.com/java/