عدد صحيح / 10 نتخلص من الآحاد

عدد صحيح % 10 نحصل على الآحاد فقط

ناتج باقي القسمة محصور بين الصفر والمقسوم عليه - 1

Sum from 1 to n = n * (n+1) / 2

Byte = 8 bits

Binary:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

وسيط بين الأعداد Decimal

التحويل من النظام السادس عشر إلى العشري نضرب في 16 ومن النظام الثماني نضرب في 8

التحويل من النظام السادس عشر إلى الثنائي نحول كل عدد إلى Nipple ومن الثماني نحول إلى ثلاثة أرقام

---

4 bit (unsigned) 0 => 15

(signed) -8 => 7

4 bit = 15 (أقصى قيمة تخزين)

5 bit = 31 (أقصى قيمة تخزين) (15*2+1)

المتمم الثنائي = two's complement

طريقة تمثيل الأعداد الموجبة والسالبة في جافا

العدد يكون 4 بت ومضاعفاتها

آخر بت خاص بالإشارة

إذا موجب 0

إذا سالب 1

كيفية إيجاد المتمم الثنائي

بنزل الأعداد زي ما هي لحد أول 1 يقابلني بينزل زي ما هو وبقلب الأعداد التي تليه.

المتمم الثنائي يشبه المعكوس الجمعي

تصنيف لغات البرمجة:

- Machine Language: Binary or Hexa
- Low Level Languages: Assembly
- Mid Level Languages: C & C++
- High Level Languages: Java, C#, Python & Java Script

كلما ارتفع المستوى كانت اللغة أسهل ولكن أبطأ

Compiled Languages vs Interpreted Languages

Compiler يأخذ ال Source Code ويفحصه بشكل كامل فإذا وجد خطأ توقف وأظهر رسالة خطأ وإذا لم يجد يقوم بإخراج ال Object Code وبعد ذلك Linker يربط ال Object Code مع ال Libraries وينتج ملف exe وال Loader يقوم بتحميله في ال memory لأجل التنفيذ.

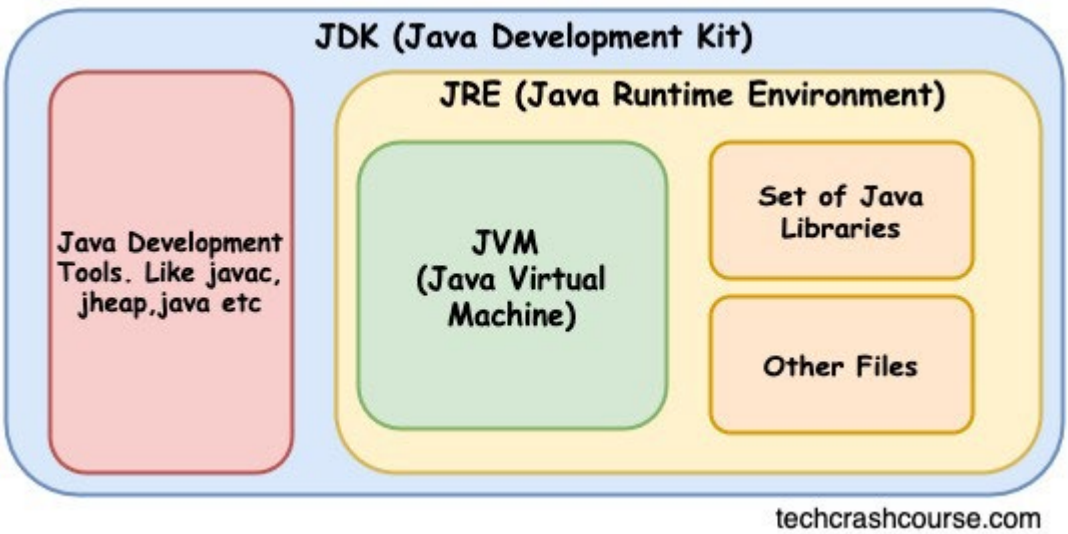
Interpreter يأخذ ال Source Code ويفحصه سطر سطر فيحول السطر الصحيح للغة الآلة لينفذ مباشرة ويقف عند السطر الخطأ.

Compiled Languages مثل C أسرع لأنها تنتج ملف exe يعمل مباشرة بمجرد تشغيله.

Assembler translates Assembly Code to Machine Code.


Byte Code
Java Virtual Machine
Native Code
*.class
Windows, Mac, ...


Java Editions
Standard Edition
Enterprise Edition
Micro Edition
Java SE
Java EE
Java ME
Desktop Applications
Server-Side Development
Mobile and Embedded Devices


JDK (Java Development Kit)
JRE (Java Runtime Environment)
Java Development Tools. Like javac, jheap, java etc
JVM (Java Virtual Machine)
Set of Java Libraries
Other Files
techcrashcourse.com

بيئة التطوير المتكاملة

IDE: Integrated Development Environment
Workspace Contains tools that help us write, debug and run code..

Library: مجموعة أكواد جاهزة

Library contains packages
package contains classes

Packages: are namespaces for our classes used to group related classes (Folder).

Package: all letters small (com.saif)

folder com => folder saif => class

root package contains subpackages

Middle Package: A subpackage that has only one subpackage and no code.

you can't have two classes with the same name inside one package.

To use class A inside another class B:
(two classes inside two different packages)
import package.A;

PascalNamingConvention: classes
Capital أول حرف من كل كلمة

camelNamingConvention: methods & Vars
Small أول كلمة ما عدا أول كلمة Capital أول حرف من كل كلمة

Java Byte Code is cross-platform.

Write once, run anywhere.

Runs on any platform has JRE.

JRE: Java Runtime Environment.

JRE: has component called JVM

JVM: Java Virtual Machine translates byte code to native code (machine code)

Binary or Hexadecimal

Compile and run java with terminal or cmd:

cd (path of java source file)

javac Main.java (Java Source File)

cd (path of src)

java package.Main (Byte Code.class)

\n: flush buffer بعد طباعة كل الجمل

println: flush buffer بعد طباعة جملة جملة

input buffer – output buffer

single line comment: //

multi line comment: /* */

Escape Sequences

\\ - \t - \n - \” - \’ - \b - \r - \uXXXX

\b: insert a backspace

\r: carriage return (إرجاع المؤشر لأول السطر)

\u: Unicode char, XXXX: Hexa-Decimal(4 NUMBERS)

Identifier: اسم المتغير

- لا يمكن استخدام كلمات محجوزة في اللغة.
- الاسم يجب ألا يبدأ برقم.
- يجب ألا يحتوى الاسم على فراغات ولا رموز خاصة ما عدا _ $.

Data Types in Java:

- Primitive: بدائي / أولي

simple values stored directly in stack.

- Reference: مرجعي

memory address in stack points to content in heap.

Ex: String – Array – Class

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 to 16 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

---

String name = new String(“Ahmed”);

String name = “Ahmed”;

scanner.nextLine();

عشان نخلص من \n اللي في Input buffer

String is immutable

عند إنشاء كائن من نوع String بقيمة معينة لا يمكن أبدًا تغيير القيمة في الذاكرة ولكن عند التعديل يتم إنشاء كائن جديد في الذاكرة من نوع String يحتوي القيمة الجديدة ثم إعادة المرجع الخاص بهذا الكائن.

String Pool is a special part of heap that store only String objs

---

Constants - الثوابت

Read only variables - لا يمكن التعديل عليها

final String BASE_URL = " ";

must be initialized when declared.

final object can't be reassigned.

Arithmetic Operators: العوامل الحسابية

+ - * / %

أولويات الحساب: الأقواس - الأسس والجذور - الضرب والقسمة - الجمع والطرح.

Increment and decrement operators:

++ --

A++ A = A + 1

A-- A = A - 1

Prefix: C = --A نعدل في القيمة المخزنة أولًا

Postfix: C = A-- نعدل في القيمة المخزنة لاحقًا

في العمليات الحسابية يتم ترقية byte & short إلى int ونحتاج Manual Casting

---

Assignment Operators:

= += -= *= /= %=

Relational Operators: عوامل النسبية أو المقارنة

== != > >= < <=

Logic Operators: العوامل المنطقية

AND: &&

OR: ||

NOT: !

&&: لو الطرف الأول false لن يفحص الآخر

||: لو الطرف الأول true لن يفحص الآخر

() → NOT → AND → OR (precedence - الأولوية)

Java Type Casting:

Widening Casting - converting a smaller type to a larger type.

Narrowing Casting - converting a larger type to a smaller type.

boolean < byte < short < char < int < long < float < double

لما تعمل كاستينج من نوع أكبر إلى نوع أصغر وتكسر الرينج بيعيد الدورة من البداية (data loss)

---

from string to int: Integer.parseInt(String)

from int to string: Integer.toString(int)

Programming Paradigms: أنماط البرمجة

- **Imperative**: تكتب خطوات تفصيلية **في ترتيب معين** لتغيير الحالة (لا تستخدم دوال)
- **Procedural**: نفس فكرة البرمجة الإجرائية، لكن الكود **منظم داخل دوال** قد تعدل حالة البرنامج أو البيانات الخارجية
- **Functional**: تركز فقط على **إرجاع ناتج** استنادًا إلى المدخلات دون التأثير على الحالة أو المتغيرات الخارجية
- **OOP**: تقسيم الكود إلى كائنات تحتوي على بيانات ووظائف
- **DECLARATIVE**: توجيه بدون خطوات تنفيذ يعني صرح بما تريد

Function Parameters: متغيرات الدالة

Arguments: القيم المرسلة

Variable Scope: نطاق (مجال) المتغير

Variable has 4 things:

- Name (identifier)
- Value
- Reference
- Scope

لا يمكن تسمية متغيرين بنفس الاسم داخل نفس النطاق

لو عندك متغيرين بنفس الاسم في الكلاس والميثود:

فالأولوية للأقرب يعني الميثود local variable rather than global

---

Random Numbers in Java

```
import java.util.Random

Random random = new Random();

int num = random.nextInt(5,10);
```

First number is inclusive

Second number is exclusive

5 => 9

```
double num = random.nextDouble();
```

0 => 0.99999999999999

Math Class

Math.PI; (= 3.14)

Math.pow(2, 3); (= 8)

Math.sqrt(100); (= 10)

Math.abs(-5); (= 5)

Math.round(5.5); (= 6)

Math.ceil(5.1); (= 6)

Math.floor(5.8); (= 5)

Math.max(n1,n2);

Math.min(n1,n2);

Conditional Statements:

condition = boolean expression

```
if (condition == true) {

} else {


}
```

```
if (condition1) {
```

```
} else if (condition2) {


}
else {


}
```

---

switch expression in java 14

```
byte number = scanner.nextByte();
String str = switch (number) {
    case 1 -> "Number is one";
    case 2 -> "Number is two";
    case 3 -> "Number is three";
    default -> "Invalid input";
};
```

```
switch (number) {
    case 1 -> System.out.println("Number is one");
    case 2 -> System.out.println("Number is two");
    case 3 -> System.out.println("Number is three");
    default -> System.out.println("Invalid input");
}
```

```
String str = switch (number) {
    case 1 -> "Number is one";
    case 2 -> {
        System.out.println("I will return a value");
        yield "Number is two";
    }
    case 3 -> "Number is three";
    default -> "Invalid input";
};
```

yield: returns a value inside a block

```
int value = 3; //multi-label case
switch (value) {
    case 1, 2, 3:
        // Executes if value is 1 or 2 or 3
```

LOOPS

```
for (initialization; condition; update) {
    // body of loop
}
```

while loop: لا نعرف عدد التكرار

```
while (condition) {
```

```
    // body of loop
}
```

```
do {
    // body of loop
} while (condition);
```

do while: تنفذ الكود مرة واحدة على الأقل حتى لو كان الشرط خاطئ

break; get out of loop

continue; move on to the next loop

تستخدم داخل الجمل الشرطية في حلقات التكرار

---

3 Types of Errors:

- Syntax Error: خطأ في قواعد الكتابة
- Run-Time Error: يحدث أثناء التشغيل
- Logical Error: الكود يعمل لكن يعطي نتائج خطأ

Ternary Operator - Short Hand if:

variable = (condition) ? expressionTrue : expressionFalse;

Ranged Loop: for each item in collection

```
for (type var : array) {
    statements using var;
```

}

Nested Loop: Loop inside loop

outer loop {Inner loop}

Nested if condition

Array can store multiple values of the same type. [ ] (Indexing Operator)

int[] nums = new int[5];

int[] nums = {1,2,3,4,5};

array.length = size of array

Arrays.sort(array);

Arrays.fill(array, "car");

2D array: Matrix of data

```
int[][] nums = new int[2][3];
int[][] nums = {{1, 2, 3}, {4, 5, 6}};
```

`Array of arrays = Every row is 1D array`

```
int[] nums1 = {1, 2, 3};
int[] nums2 = {4, 5, 6};
int[][] nums3 = {nums1, nums2};
for (int[] i : nums3) {
   for (int j : i) {
      System.out.print(j + " ");
   }
```

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

Bitwise and operator: &

Dcimal to Binary then anding bit by bit.

Bitwise or operator: |

Dcimal to Binary then oring bit by bit.

---

var: auto data type

used only with methods

String Methods

int str.length();

str.charAt(0);

str.indexOf('E')

str.lastindexOf('E') case sensitive

str.toUpperCase();

str.toLowerCase();

str.trim(); Removes whitespace from both ends

str.replace("", "");

str.isEmpty();

str.contains(" "); true if contains whitespace

boolean str.equals(str2);

str.equalsIgnoreCase(str2);

Str.endsWith(" ");

Str.substring(begin,end)

begin: inclusive - end: exclusive

substring(5); From index 5 to the end of the string

Str.concat(" ");

"Hello" + " Ahmed" = "Hello Ahmed"

Format output | التنسيق في لغة الجافا

"\u202B": right to left

"\u202A": left to right

System.out.printf()

Placeholder Form:

%[flags][width][.precision]conversion

Conversion: (Required)

%: percent

n: line break

b: Boolean

d: decimal integer

f: float / double

s: string

c: char

---

flags:

+: إشارة موجبة قبل الرقم الموجب

-: محاذاة لليسار (width)

لو كتبت ال width من غير - تبقى محاذاة لليمين

0: يضيف أصفار مكان الفراغات (width)

,: يفصل بين مجموعات الرقم الكبير

لا يستخدمان معًا - | 0

space char : مسافة قبل العدد الموجب ليحاذي السالب

(: الرقم السالب بين قوسين

---

width: عرض الحقل (الحد الأدنى)

.precision: .number of decimal digits

ومع string عدد الحروف المراد طباعتها

"K&R Style" (Kernighan & Ritchie Style)

كتابة القوس الافتتاحي ع نفس السطر وكذلك else and else if

String x = """

"""";

Text Block (Triple double quotes)

Function inside a class called method.

Method: a block of reusable code that is executed when it's called.

Access modifier for class, methods and attributes.

Return Type & Parameter: Primitive or Reference

int[] – String[]

void: procedure (Return nothing)

Method Signature (Unique Identification):

1- Name
2- Number of parameters
3- Type of parameters

4- Order of parameters

method overloading: multiple methods with the same name, but different signatures.

Java is always pass-by-value.

Variable Arguments (Varargs) in Java

Allow method to accept variable number of arguments. (or no argument)

Java will treat these values like a list.

No need for overloaded methods.

static void display(String... name)

Read and Write Files

FileWriter: Good for small or medium-sized text files.

BufferReader + FileReader: Best for reading text files line-by-line.

Create & Write

```
import java.io.*;

try (FileWriter writer = new FileWriter("C:\\Users\\Saif\\Desktop\\test.txt")){
   System.out.println("File has been created");
   writer.write("Hello World\nAhmed Saif");
}
```

```
catch (FileNotFoundException e) {
   System.out.println("Error while creating file");
}
catch (IOException e) {
   throw new RuntimeException(e);
}
```

Read File

```
import java.io.*;

try (BufferedReader reader = new BufferedReader(new
FileReader("C:\\Users\\Saif\\Desktop\\test.txt"))){
   System.out.println("File is Found");
   String line;
   while ((line = reader.readLine()) != null) {
      System.out.println(line);
   }
}
catch (FileNotFoundException e) {
   System.out.println("File is not Found");
}
catch (IOException e) {
   throw new RuntimeException(e);
}
```

OOP

Programming Paradigm

Class: A blueprint for creating objects.

A group of attributes (fields) and methods.

Makes code structured, reusable and scalable.

Allows me to create my own data type.

Field can be primitive or reference variable.

Object: An instance of a class

Objects have the same fields of class but with different values.

Class	Object
Logical Construct	Physical Reality (in memory)

Access instance variables with (. dot operator)

Student student; //declaration null بیشاور علی
student = new Student(); //initialization

new: dynamic memory allocation.

Memory Segments: Stack – Heap –

Method Area – Program Count Register

Stack: Allocated in compile time

Heap: Allocated in Run time

Stack: primitive variables and variables that store references to objects

Heap: object with its contents

All fields inside class are stored in heap

stack is faster than heap

In Java, references are declared at compile time, while objects are created (initialized) at runtime.

The JVM does this automatically by default:

- initializes uninitialized fields with 0, 0.0, null, false when you instantiate an object.
- initializes objects inside class or array with null.

الجافا بتضيف Default Constructor أوتوماتيك لو مكتبتش أي Constructor بنفسي وده بيكون فاضي وبدون بارامترز.

A constructor is a special method that is called when an object of a class is created.

كأن ال Constructor هو البوابة اللي بيدخل منها الأوبجكت للكلاس.

new Student(); default constructor is called.

new: Dynamic Memory Allocation in heap.

1) constructor has the same name as the class
2) constructor has no return type

this keyword: refers to the current object

- we must use it if the parameters and fields have the same names.
- You can call constructor from another constructor this();

(this replaced internally with the name of the class = constructor)

Constructors can be overloaded.

Wrapper classes provide a way to use primitive data types as objects. (غلاف للنوع البدائي)

(Byte – Short – Integer – Long – Float – Double – Boolean – Character)

```
Integer num = 5; //auto boxing int to Integer
System.out.println(num); //auto unboxing Integer to int
```

wrapper classes are final (يعني مينفعش تورث منها)

wrapper class object is immutable

immutable: غير قابل للتغيير

يمكن تغيير ال reference ولكن لا يمكن تعديل المحتوى

Static: object independent

Belongs to the class not the object.

Static Method = Class Method:

- Deals only with static members
- Doesn't have this pointer

Static Attributes: Shared attributes among all objects. Class.field or method (Access)

Static methods are loaded once into the Method Area when the class is loaded, and they are not duplicated for each instance. This reduces memory usage and improves performance.

تقدر تأكسس من خلال الأوبجكت بس الأفضل من خلال الكلاس نفسه

In static methods you can create objects to access non-static members.

الميثود لما بنستدعيها جوا ميثود تانية كأنها قاعدة في بيتها وبتطلب منها أوبجكت لو هي non-static

static block: used for static initialization, it executes one time automatically when the class is loaded into memory for the first time.

```
static {

}
```

Nested Classes:

- Inner (Non Static)
- Static

Unlike top-level classes, Nested classes can be Static.
Non-static nested classes are also known as Inner classes.

الكلاس الداخلي كلاس وله ال members الخاصة به والكلاس الخارجي كلاس آخر وله ال members الخاصة به.. ولكن الكلاس الداخلي يعتبر member

في الكلاس الخارجي وبالتالي يمكنه الوصول ل members الكلاس الخارجي بشكل مباشر والعكس غير صحيح.
وبديهيًا يمكنك إنشاء كائن من الكلاس الخارجي داخل الكلاس الداخلي والعكس صحيح.

(الرابط) في الـ Main:

You can create object directly from outer class but you can't create it directly from inner class as it's dependent on outer class.

```
Outer o =  new Outer();
Outer.Inner i = o.new Inner();
```

مساحة جديدة من نوع inner واربطها بالكائن o فبيخزن فيها Reference للكائن o

إزاي أوصل لل Outer class attributes من خلال الكائن i مباشرة؟

i.access(); هستدعي

```
void access() {
   Outer.this.outerY = 5; (ميثود داخل الكلاس الداخلي)
}
```

Outer.this: refers to the Outer object

associated with the current object.

Static Class: independent of object

غير معتمد (مستقل) على أوبجكت الكلاس الخارجي

You can instantiate object directly from it

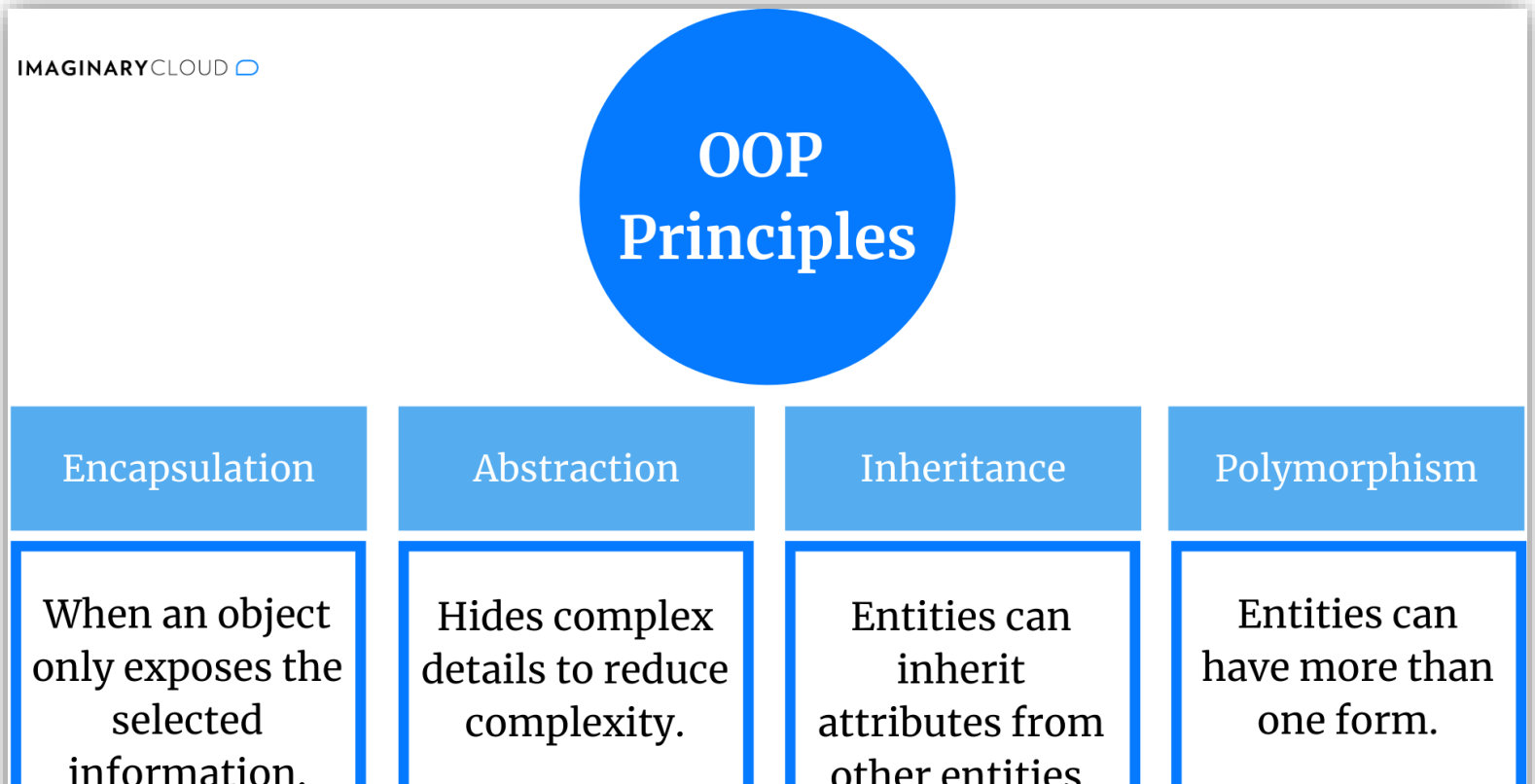
```
Outer.Inner i = new Outer.Inner(); //(in Main)
```

static class can access only the static members of the outer class.

Java singleton class is a class that can have only one object. If you instantiate more objects, they will all point to the same reference.

```
Singleton object = Singleton.getInstance();
class Singleton {
    private Singleton() {


    }
   private static Singleton instance;
    public static Singleton getInstance() {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}
```



Inheritance

Superclass (Parent) ⟶ Subclass (Child)

To inherit from a class, use the extends keyword.

```
class Car extends Vehicle {

}
```

subclass inherits attributes and methods from the superclass in addition to its own members.

It is useful for code reusability. **(Remove duplication from the code)**

Vehicle car = new Car();

Instance car can only access Vehicle members.

Car car = new Vehicle(); //Error

Car has its attributes that do not exist in Vehicle.

Vehicle(): can't initialize attributes of Car

Every class in java extends Object Class. **(Internally)**

لو عندك ميثود في الكلاس الأب فيها بارامتر reference من نوع الأب تقدر ت pass ليها reference من نوع الابن و هتستقبله عادي.

---

super keyword - refers to the superclass

super(); //constructor of the subclass calling the constructor of its parent (superclass).

Must be the first line in the body of constructor.

- الكلاس الأب لا يهمه محتوى الكلاس الابن بينما الابن يهمه محتوى الأب لأنه وارث منه فلازم الأول يعمل لمحتواه تهيئة عن طريق super(); فلازم تكون أول سطر في كل Constructor.
- إذا لم تكتب super(); يدويًا فالجافا تستخدمها ضمنيًا لاستدعاء Default Constructor (with no parameters) فإذا لم تجده تعطيك Error.

super is used to access only non-static fields of the Parent class.

الكومبايلر هيسمح بس الأفضل عدم استخدامها لأن معناها (اذهب إلى نسخة الكائن من هذا المتغير في الكلاس الأب) واستخدام Parent.field

لو الكلاس الابن والكلاس الأب عندهم field بنفس الاسم وأنا واقف في الكلاس الابن وعايز access field بتاع الأب:

super.field (non-static)

Parent.field (static) Parent → اسم الكلاس الأب

Types of inheritance

. Single inheritance
. Multilevel inheritance
. Multiple inheritance (in C++ not Java)
. Hierarchical inheritance
. Hybrid inheritance

- Single inheritance: a class is allowed to inherit from only one class.
- Multilevel inheritance:

SubClass2 inherits members from SubClass1 & SuperClass.

- Multiple inheritance: in c++ a class can inherit from more than one class.

لو كلاس وارث من كلاسين عندهم متغيرين بنفس الاسم هتحصل مشكلة !

الكلاس الابن هيحتار ولذلك الجافا لا تدعم الوراثة المتعددة. (interface fix)

- Hierarchical inheritance: a single superclass is inherited by many subclasses.
- Hybrid inheritance: is implemented by combining more than one type of inheritance. ***(combination of single and multiple inheritance)***

In Java, we can achieve hybrid inheritance only through Interfaces.

D وارث نستختين من نفس ال member نسخة عن طريق B ونسخة عن طريق C فيا ترى لما يحب يستدعيه هيوصل لأي نسخة فيهما

A
/ \
B C **Diamond Problem in multiple inheritance**
\ /
D

Polymorphism

having many forms

the ability of an object to take on many forms

Representing the same thing in multiple ways

EX: the same method signature inside inherited classes with different implementations.

"The method that gets called is determined at runtime based on the actual object type, not the reference type."

Animal a = new Dog();

Animal: reference type - Dog: actual object type

what you can access depends on reference type

Types of polymorphism

1. **Compile-Time Polymorphism (Static)**
2. **Runtime Polymorphism (Dynamic)**

Compile-Time Polymorphism (Static) via method overloading

method overloading: multiple methods with the same name, but different Signatures. (EX: Constructor Overloading)

The compiler picks the correct method during compile time based on the arguments.

Runtime Polymorphism (Dynamic) via method overriding

method overriding:

- methods have the same signature, access modifier and return type but different implementation.
- a subclass provides a specific implementation for a method defined in its superclass.

The method that gets called is determined at runtime based on the actual object type, not the reference type.

@override annotation (بيان وصفي للمترجم – أعد تعريف)

Car extends Vehicle

Vehicle obj = new Car(); //Upcasting(A Car is a Vehicle)

obj.access(); يأكسس الميثود بتاع الأب وينفذ نسخة الابن

- ✓ Upcasting is the typecasting of a child object to a parent reference. (Automatic / Implicit)

  Any object of a child class will always satisfy the contract of its parent class.

- ✓ Downcasting is the typecasting of a parent object to a child reference. (Manual / Explicit) (Do Upcasting first)

  Car obj1 = (Car) obj;

reference type (Compiler handles it) in compile time – object type (JVM) in run time

(لازم يكون فيه علاقة وراثة بين الكلاسين)

Dynamic method dispatch: is the mechanism by which a call to an overridden method is resolved at run time.

الربط الديناميكي للدالة: الآلية التي عن طريقها يتم حسم استدعاء ميثود معاد تعريفها في وقت التشغيل.

You can override toString() method of Object class.

---

final keyword in java

final method: [Access Modifier] [static] [final] [return type]

It can't be overridden by a subclass.

Early Binding = الربط المبكر

يعني المترجم بيعرف نسخة الدالة اللي هيتم استدعاؤها في compile time

final Class: [Access Modifier] [final]

It can't be extended by a subclass.

Prevent inheritance therefore prevent method overriding

عشان تطبق مبدأ ال Polymorphism لازم يكون:

Reference Type(Parent) != Actual Object Type(Child)

كدا Override بدون تطبيق ال Polymorphism

Reference Type(Child) = Actual Object Type(Child)

---

static method cannot be overridden

static method is considered early binding

early binding: the method to be called is determined at compile time.

فأصبح المترجم بيعرف إنه هيستدعيها في ال Compile time بناءً على ال Reference type وده ميعتبرش Override وأصلًا ال static معناها إن الميثود نسخة خاصة بالكلاس تحديدًا فمفيش داعي لل Override

Method Hiding: the same static method in subclass with different implementation.

Encapsulation

wrapping up or binding fields and methods implementation into a single unit. (Implementation level)

EX: A bank provides access to customers through some methods and rest of the data is hidden to protect from outside world.

EX: Make fields private and manipulating them with setters and getters.

Abstraction

hiding certain details and showing only essential information to the user. (Design level)

التجريد: بتستخدم الحاجة ببساطة وأنت مش عارف اللي بيحصل خلف الكواليس

EX: A car driver only needs to know how to drive it not how its engine and internal components work.

Abstraction focuses on the external working

and the process of gaining information.

Encapsulation focuses on the internal working and the process of containing information. (getter and setter)

We can implement abstraction using abstract class and interfaces.

encapsulation can be implemented using by access modifier and nested classes.

Data Hiding: The internal data of an object is hidden from the outside world, preventing direct access.

Data Hiding is subprocess of Encapsulation.

Data Hiding is achieved by Encapsulation.

Access Modifiers

- private: accessible only within the declaring class.
- public: accessible from everywhere in the program.
- default / package-private: (no modifier) accessible only in the same package.

- protected: accessible in the same package and subclasses. *(even in different packages)*

A top-level class has only two possible access levels: **package-private** or **public**.

protected: members مصممة لل

private: doesn't make sense

في النوع **protected** الأوبجكت من ال **Subclass** بيحاول يأكسسه فال **Superclass** بيتأكد الأول إنه وارث منه بس بيسأله أضمن منين إنك داخل ال **body** بتاعك **subclass** وأنا معرفش محتواه فالكومبايلر بيبص على ال **reference type** بتاع الأوبجكت وبيتأكد إذا كان موجود داخل ال **Subclass** بتاعه.

Whet to use

private: sensitive data and internal helper methods

default: package-scoped utilities

protected: inheritance-based designs like framework extensions

public: API endpoints and service classes

Access modifier restrictions in decreasing order:

1. **private**

- **default**
- **protected**
- **public**

If you are overriding any method, the overriding method must not have a more restrictive access modifier.

---

package types: user defined – built-in

Built-in Packages: (java main folder)

- java.lang: Contains language support classes(classes which defines primitive data types, math operations). This package is automatically imported.
- java.io: Contains classes for supporting input / output operations.
- java.util: Contains utility classes which implement data structures and Data / Time operations.
- java.applet: Contains classes for creating Applets.

- java.awt: Contain classes for implementing the components for graphical user interfaces. (Not Recommended)
- java.net: Contain classes for supporting networking operations.

Object Class

is in java.lang

The Object class acts as a root of the inheritance hierarchy in any Java Program.

Every class extends the object class internally.

The Object class provides several methods such as toString(), equals() and hashCode(). (@Override)

- The toString() provides a String representation of an object.
- hashCode(): For every object, JVM generates a unique number which is a hashcode, it converts the internal address of the object to an integer by using an algorithm.
- obj1.equals(obj2): It is recommended to override this method to define custom equality conditions.

- getClass(): is used to get the actual runtime class of the object.

Instantiation = Declaration + Initialization

instanceof Keyword in Java

used for checking if a reference variable contains a given object type or not.

Child childObj = new Child();

childObj instanceof Child (true)

childObj instanceof Object (true)

childObj instanceof Parent (true)

Parent pObj = new Child();

Child childObj = (Child) pObj;

childObj instanceof Child (true)

pObj instanceof Parent (true)

Parent pObj = new Parent();

pObj instanceof Child (false);

Parent pObj = null;

pObj instanceof Parent (false)

Abstract Class

parent class tells us what to do not how to do it (General Form)

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

if A is an abstract Class

A obj = new A();//Can't be instantiated (A obj;)

Abstract Method: can only be used in an abstract class, and it doesn't have a body. The body is provided by the subclass. (Must be overridden)

An abstract class can have both abstract and regular methods.

[Access Modifier]...[abstract keyword][Return Type]

abstract void test2();

If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract methods.

You can define constructor and fields in the abstract class.

تقدر تعمل abstract class من غير أي ميثود abstract

Abstract (class | method) cannot be final.

abstract method: must be overridden

static method: cannot be overridden

You can't create static abstract method but you can create static method inside abstract class.

Interface and abstract class are used to group related methods with empty bodies.

Interfaces

interface keyword rather than class

used to achieve abstraction and multiple inheritance in Java

interface: is an abstract class that must be implemented by another class. (implements keyword)

لأنك مقتصر على عمل implement للميثودز بس ومفيش وراثة.

On implementation of an interface, you must override all of its abstract methods.

Types of interface methods:

- abstract
- default
- static
- private

interface methods are by default abstract and public

interface attributes are by default public, static and final

An interface cannot contain a constructor, but an abstract class can.

لأنها مصممة لتكون مجرد واجهة أو خطة مش قاعدة بناء.

fields are final so they don't need a constructor.

abstract class can implement interface but interface can't extend any class. (only extend similar interfaces)

abstract class can extend and implement.

A class can implement multiple interfaces separated with a comma.
An interface can extend multiple interfaces separated with a comma.

"Multiple inheritance achieved with interfaces"

لو فيه ميثود متكررة بنفس الاسم في ال interfaces فكأنك عامل Override ليهم كلهم يعني مفيش تعارض لأن الكلاس هينفذ نسخته.

العلاقة في الواجهة	العلاقة في الوراثة
Can-do	is-a

Interface name should be a descriptive adjective or noun like Runnable and Map. (describe a capability or role)

As we said you can't create object from interface, interface can be reference type only.

Remember: what you can access depends on reference type but what version is implemented depends on object type.

More than one class can implement the same interface.

A default method in interface is a public method with a default body that gets executed if not overridden.

```
default void doThat() {
   System.out.println("Hello in doing That");
}
```

primary motivation: default body for all classes that implement the interface.

A static method inside an interface is not inherited by implementing classes, so it can only be called using the interface name.

When an interface extends another interface, any class implementing the child interface must implement (Override) all methods from both interfaces.

A Nested Interface is an interface declared inside a class or another interface.

class A implements OuterInterface.NestedInterface
class A implements OuterClass.NestedInterface

- A nested interface inside a class can be public, protected, package-private (default) or private.

لأنها member عادي

- A nested interface inside another interface is implicitly public static.

لأنها public member عادي وstatic لأنها ليست مرتبطة بحالة الكائن

- A top-level interface (not nested) can only be public or package-private (default).

لأنك هتأكسسها من كيان خارجي وليس داخلي وprotected متنفعش مع top-level

---

annotation

بيان وصفي للمترجم

metadata for the compiler

Annotations are special (Interfaces) notes that start with the @ symbol. they provide extra information to the compiler.
They can change the way a program is treated by the compiler.

java.lang.annotation.Annotation

Built-In Annotation Custom Annotation

Some of the most commonly used built-in annotations:

@Override	**Indicates that a method overrides a method in a superclass**
@Deprecated	**Marks a method or class as outdated or discouraged from use**
@SuppressWarnings ("unchecked")	**Tells the compiler to ignore certain warnings**

The Java Collections Framework provides a set of interfaces (like List, Set, and Map) and a set of classes (ArrayList, HashSet, HashMap, etc.) that implement those interfaces.

All of these are part of the java.util package.

A **framework** is a structured base for application development that provides tools and elements to speed up the process, allowing you to avoid building everything from scratch. It acts as a template that can be used and modified to meet project requirements.

Generics means parameterized types.

The <> is called the diamond operator.

Generics allow you to write classes, interfaces, and methods that work with different data types, without having to specify the exact type in advance.

This makes your code more flexible, reusable, and type-safe.

type safe بتخليها تقبل نوع بيانات محدد

<T> type parameter it's a placeholder for any data type

<Integer> type argument

ArrayList<Integer> al = new ArrayList<~~Integer~~>();

You can declare multiple type parameters

بنفس ترتيب الحروف <T, U, V, W>

Java Collections like ArrayList and HashMap use generics internally.

Generics Work Only with Reference Types

The actual object type cannot be parameterized when you instantiate an object.

T obj = new T(); //Error

The instanceof operator in Java works at runtime so it can't be used with parameterized types.

you can declare an array of a parameterized type, but you cannot directly initialize it.

(why?) the java compiler erases all type parameters, you can't verify which parameterized type for a generic type is being used at runtime. (type erasure)

You can't declare static fields whose types are type parameters because static fields can't be shared among different classes.

A class cannot have two overloaded methods that will have the same signature after type erasure.

Java has an abstract class called java.lang.Number

It is the **superclass** of all numeric wrapper classes like:

- Integer
- Double
- Float

Bounded Types

You can use the extends keyword to limit the types a generic class or method can accept.

class Stats<T extends Number>

T is a Number

the type must be a subclass of Number

void getList (List<Number> list)

void getList (List<T extends Number> list)

في الميثود الأولى type argument عبارة عن Number فكلاس List هيستقبل من نوع Number نفسه بس بينما في الثانية هيستقبل أي subclass من Number عادي.

Type Parameter Naming Conventions:

- T: Type (نوع بيانات عام)

- E: Element (عنصر داخل كوليكشن)
- K: Key (مفتاح في الماب)
- V: Value (القيمة المقابلة للمفتاح)
- N: Number (بيرمز لرقم)

Comparing Objects

List.contains(Object o) uses .equals() to check if any element in the list is equal to the given object. //Override equals()

Comparable is interface with generic type

```
class Student implements Comparable<Student> {
    @Override
   public int compareTo(Student obj) {
      int diff = (int)(this.marks - obj.marks);
      return diff;
   }
}
```

Functional Interface is an interface that contains exactly one abstract method.

It can have any number of default, static or

private methods.

Single Abstract Method Interface (SAM), it helps achieve functional programming approach.

A functional interface can't contain any Object's method because all implementing classes inherit from Object.

يعني مش داخلين في الحساب لازم تعمل one abstract method بايدك.

A functional interface can extend another interface only if it does not have any abstract method.

Anonymous Inner Class

A nested class without a name that is declared and instantiated at the same time, used if we need to use a local class only once. (in one place)

Anonymous بيستخدم في نفس المكان ولو حبيت تعمل **Implementation** جديد يبقى لازم تعمل **Class** جديد

Local Class: inner class inside method or block.

```
Test t = new Test() {
  // fields and methods
};
```

- ✓ Overriding methods of a class or interface, without having to actually subclass a class.

Anonymous inner classes are created via two ways:

1. Class (abstract or concrete)
2. Interface

- Anonymous class extends the superclass internally and can have its own members.
- Object cannot access the members of anonymous class.

  A obj = new Main$1(); //JVM generates hidden class Main$1

  reference type: A

  can only access members of A
- Anonymous inner class can extend a class or can implement only one interface but not both at a time.

- Anonymous inner class doesn't have a constructor.

```
interface Age {
   int x = 21;
   void getAge();
}
public static void main(String[] args){
      Age oj1 = new Age() { //oj1 object of anonymous class
         @Override
         public void getAge(){
            System.out.print("Age is " + x);
         }
      };
}
```

Anonymous inner class as an argument

```
abstract class Engine {
  public abstract void engineType();
}
class Vehicle {
  public void transport(Engine e) { // = new Tester$1();
    e.engineType();    //Engine e = new Tester$1();
  }
}
public class Tester {
```

```
  public static void main(String args[]) {
    Vehicle v = new Vehicle();
    v.transport(new Engine() { // = new Tester$1()
      @Override
      public void engineType() { //Anonymous Inner Class
        System.out.println("Turbo Engine");
      }
    });
  }
}
```

Lambda Expressions were added in Java 8

Lambda expressions let you express instances of functional interfaces more compactly.

A lambda expression provides a concise way to represent an anonymous class that implements a functional interface.

Lambda generates hidden object at runtime which implements the functional interface and overrides SAM, We interact with it through the interface reference.

Conciseness – Readability – Functional Programming

لامدا أسرع وأكثر كفاءة في الذاكرة لأنها لا تنشئ **anonymous class** ومقتصرة على تنفيذ الميثود وبالتالي لا يمكنها الوصول ل **fields of interface**.

Works only with functional interface (one abstract method – No Confusion).

(parameters) -> {body}

Arrow token

- ✓ **You can omit parameters types**
- ✓ **You can omit parentheses in case of a single parameter**

Types of lambda parameters:

1. Lambda with No parameters

```
Runnable runnable = () -> System.out.println("Hello, World!");
```

2. Lambda with a Single parameter

```
Greeting greeting = name -> System.out.println("Hello, " + name);
```

3. Lambda with Multiple parameters

```
Add obj = (a,b) -> System.out.println(a+b);
```

The compiler infers the return type of a lambda based on the SAM of the functional interface.

```
Add obj = (a, b) -> { // (a, b) -> a + b; (Implicit Return)
   int x = a + b;
   return x;
}; // when you have a single statement return should be removed.
```

The forEach() method takes a lambda expression or a method reference as a parameter and executes it for each element of the list.

```
ArrayList<Integer> al = new ArrayList<Integer>();
al.forEach(n -> {
        if (n % 2 == 0)
           System.out.println(n);
});
```

The forEach method takes a lambda expression as an argument, and that lambda must implement the Consumer functional interface.

In General a lambda can be passed as an argument to a method that accepts a functional interface.

arr.forEach(instance storing lambda expression);

Method References

- Reference to a static method: Math::sqrt
- Reference to an instance method of a particular object: str::length (only str object)
- Reference to an instance method of an arbitrary object of a particular type: Person::sayHello (for any Person object)
  String::toUpperCase

```
ArrayList<String> list = new ArrayList<>();
        list.add("Ahmed");
        list.add("Ali");
        list.forEach(String::toUpperCase);
```

ال method reference اختصار ل Lambda في حالة إنك بتستدعي ميثود موجودة بالفعل

Legacy Functional Interfaces

Early Before Java 8

annotated with @FunctionalInterface

Ready to use with lambda and method reference

- Runnable: This interface only contains the run() method.
- Comparable: This interface only contains the compareTo() method.
- ActionListener: This interface only contains the actionPerformed() method.
- Callable: This interface only contains the call() method.

Built in Functional Interfaces

Java 8 provides many general-purpose functional

interfaces in the java.util.function package which we can use to define lambda expressions. They all fall into four categories:

1. Supplier: T get(); //(abstract method)

doesn't take any input or argument and returns a single output

```
Supplier<Integer> supplier = () -> 100;

System.out.println(supplier.get());
```

In case you want a primitive, rather than an object, back from the supplier.

BooleanSupplier, DoubleSupplier, LongSupplier and IntSupplier

```
Supplier<Integer> supplier = () -> -5; //return Integer object

IntSupplier intSupplier = () -> -5; //return primitive int
```

2. Consumer: void accept(T t);

accepts only one argument and returns no result.
(has no return value)

```
Consumer<Integer> consumer = a -> System.out.println(a);
consumer.accept(3);
```

There are also functional variants of the Consumer DoubleConsumer, IntConsumer, BiConsumer and LongConsumer.
BiConsumer: accepts two arguments and returns no result.
void accept(T t, U u); //abstract method

```
BiConsumer<Integer, String> biConsumer = (a,

b) -> System.out.println(a + b);

biConsumer.accept(125, " Hello"); //Two Arguments and no return
```

There are also BiPredicate and BiFunction but no BiSupplier

3. Predicate: boolean test(T t);

takes a value, does a logical test, and returns a boolean value.

There are IntPredicate, DoublePredicate and LongPredicate accept only primitive data types or values as arguments.

It is commonly used for filtering operations in streams.

```
Predicate<String> predicate = value ->  value != null;
System.out.println(predicate.test("hello")); //true
```

4. Function: R apply(T t);//R: Result Type

receives only a single argument and returns a result after the required processing.

IntFunction, DoubleFunction and LongFunction. (different versions used in primitive types)

LongFunction: receives one argument of type long and returns a result.

```
Function<Integer,Long> func = (value) -> (long)(value * value); //Arithmetic
Expression (manual conversion is required)
System.out.println(func.apply(10));
```

Some Functional Interfaces to suit specific needs

Functional Interface	Description	Method

UnaryOperator<T>	This is a special case of Function, where input and output types are the same.	T apply(T t)
BinaryOperator<T>	This is a special case of BiFunction, where input and output types are the same.	T apply(T t1, T t2)
LongToDoubleFunction	This is a special case of LongFunction returns a double result.	double applyAsDouble(long value);
ObjIntConsumer<T>	This is a special case of BiConsumer.	void accept(T t, int value);
BooleanSupplier	special case of Supplier	boolean getASBoolean();

returns boolean-valued result (true / false)

```
ObjIntConsumer<Ostora> consumer = (ostora, value) ->
System.out.println(ostora.x + value);
consumer.accept(new Ostora(), 10); //Output: 15

class Ostora {
   int x = 5;
}
```

Coupling: اقتران أو ترابط

The level of dependency each class or component in application has on another.

Tight Coupling (ترابط محكم) means features of different classes and objects have high dependence on one another, whereas Loose Coupling (ترابط رخو/غير محكم) means components have very low or no dependence on one another.

In tight coupling, if the implementation of one class changes, the dependent class might also need to be changed. It makes Java code less flexible and more difficult to maintain, especially in large-scale applications.

Loose coupling is a design goal to reduce the interdependencies between components of a system.
In loose coupling, classes communicate through separate interfaces.

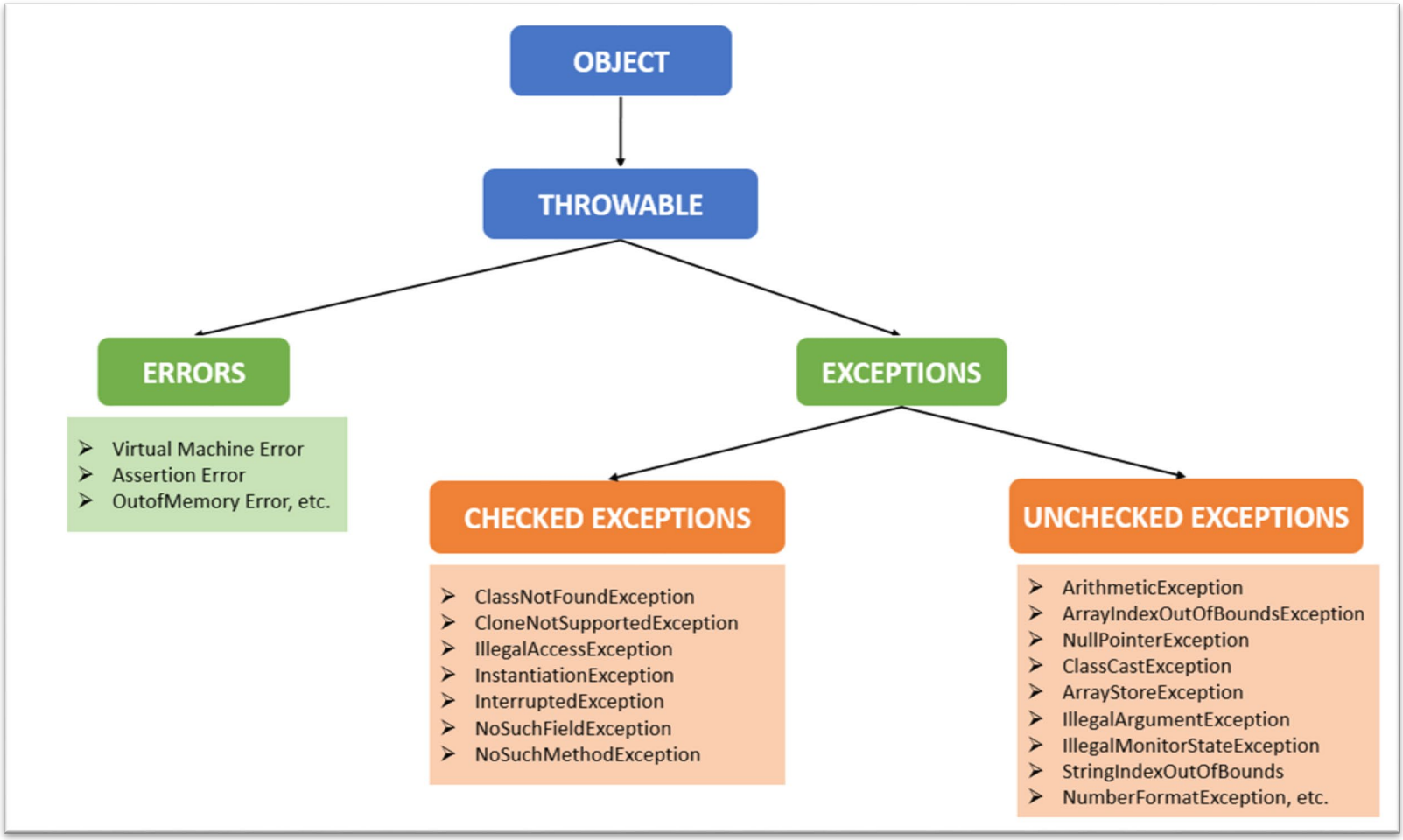
Exception Handling – التعامل مع الاستثناءات

Runtime Error: is an error that occurs during running or execution of a program after being successfully compiled.
Leads to unexpected behavior or crash in the form of an Exception or Error.

Error	Exception
Subclass of java.lang.Throwable	Subclass of java.lang.Throwable
Unchecked by the compiler	Can be checked or unchecked
Indicates serious problems that a reasonable application should not try to catch	indicates conditions that a reasonable application might try to catch and handle

the application is unrecoverable	the application is recoverable
Caused by the JVM running out of resources, other system failures or hardware	Caused by external conditions like file not found, network issues, or invalid user input
Examples: OutOfMemoryError, StackOverFlowError	Examples: IOException, NullPointerException

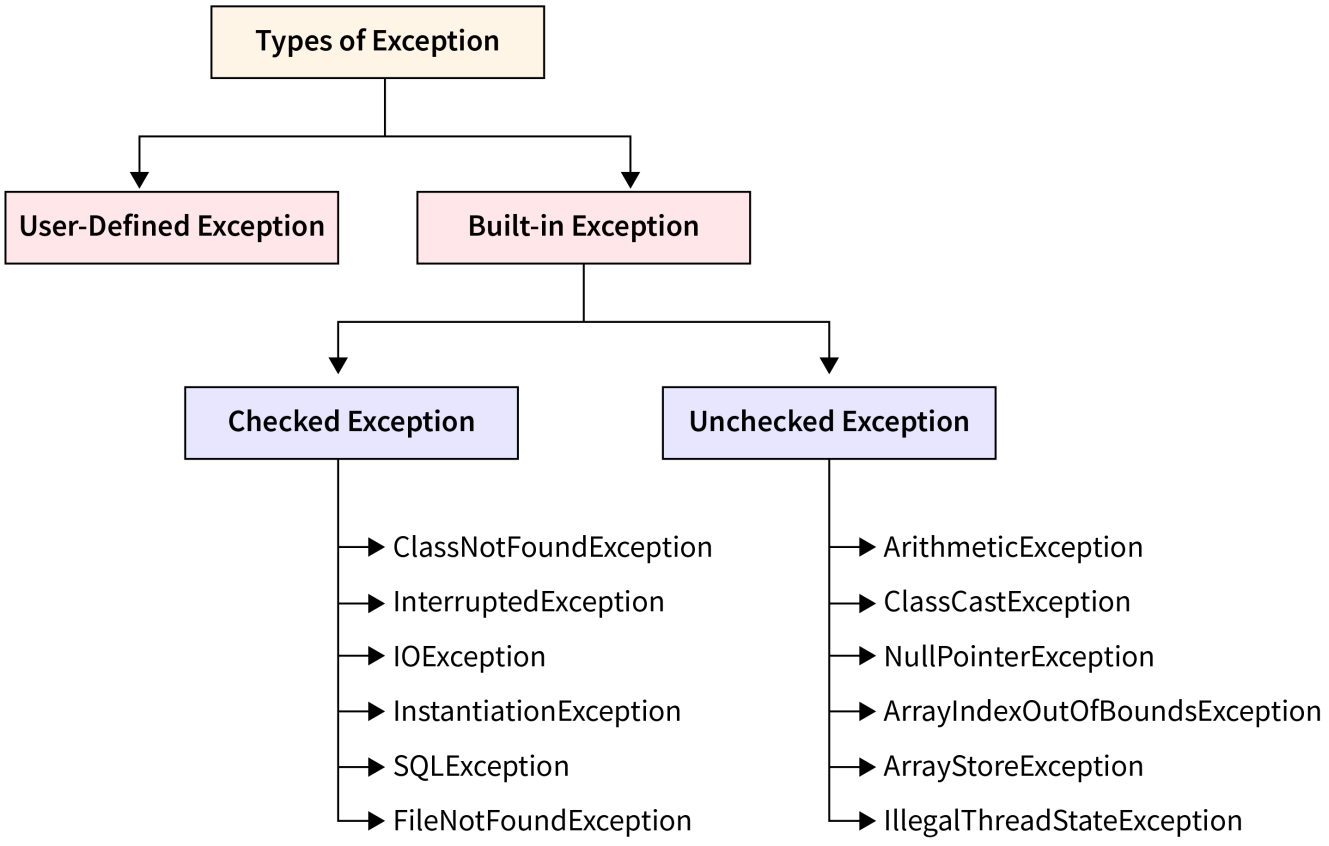
An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. Exception Handling is a mechanism to handle and manage runtime errors, allowing a program to continue running or terminate gracefully.



It separates error-handling code from the main program logic, making the code more robust and readable.

1. Built-in Exceptions: pre-defined exception classes provided by java to handle common errors during program execution.

- Checked Exception
- Unchecked Exception



Checked Exceptions: are called compile-time exceptions because they are checked at the compile-time by the compiler. **(extend Exception)**

- ClassNotFoundException: Throws when the program tries to load a class at runtime but the class is not found because it's belong not present in the correct location or it is missing from the project.
- InterruptedException: Thrown when a thread is paused and another thread interrupts it.
- IOException: Throws when input/output operation fails.
- InstantiationException: Thrown when the program tries to create an object of a class but fails because the class is abstract, an interface or has no default constructor.
- SQLException: Throws when there is an error with the database.
- FileNotFoundException: Thrown when the program tries to open a file that does not exist.
- IllegalAccessException: Thrown when an application tries to reflectively create an instance or access a field or method that it doesn't have the right to access.

Unchecked exceptions: The compiler will not check them at compile-time. **(extend RunTimeException directly or indirectly)**

مش بيدي compile error والكود بيترجم ويتنفذ ولكن لو حصل الاستثناء البرنامج يقع في وقت التشغيل بينما checked بيجبرك تعالجه قبل ما تعمل Run وتحصل مشكلة.

- ArithmeticException: It is thrown when there is an illegal math operation.
- ClassCastException: It is thrown when we try to cast an object to an incompatible class type.

- NullPointerException: It is thrown when we try to use a null object (e.g. accessing its methods or fields).
- ArrayIndexOutOfBoundsException: This occurs when we try to access an array element with an invalid index.
- ArrayStoreException: This happens when we store an object of the wrong type in an array.
- IllegalThreadStateException: It is thrown when a thread operation is not allowed in its current state.

2. User-Defined Exception: Users can also create custom exceptions to represent specific error types.

Methods to Print the Exception Information

1. printStackTrace(): **Prints the stack trace of the exception, including the name, message and location of the error.**
2. toString(): **Returns the exception name and message only. (Overridden in Throwable class)**
3. getMessage(): **Returns the description of the exception (message only).**

How does JVM handle an exception?

When an exception occurs the JVM creates an exception object containing information about the exception and
handing it to the runtime system which is called "Throwing an exception".
There might be a list of methods that had been called to get to the method where an exception occurred. This ordered list of methods is called call stack.

- The runtime system searches the call stack for an exception handler.
- It starts searching from the method where the exception occurred and proceeds through the call stack in reverse order. (Unwinding the call stack)
- If an appropriate handler is found, the runtime system passes the exception to the handler.
- If no handler is found, the default exception handler terminates the program and prints the stack trace.

Unwinding the call stack: التراجع التدريجي ل (فك) مكدس الاستدعاء

Stack Trace: تتبع مكدس الاستدعاء

بيعرض اسم الاستثناء والرسالة ومكان الخطأ عبارة عن الترتيب العكسي لمكدس الاستدعاء

Run-Time System = JRE ل **القلب النابض**

A try-catch block in Java is a mechanism to handle exception:

- JVM starts executing the statements inside the try block sequentially. If it completes all statements without an exception being thrown, it skips the catch blocks entirely and proceeds directly to the finally block (if exists).
- When an exception is thrown inside the try block, the JVM immediately stops execution there and looks for a catch block whose parameter type matches the type of the thrown exception object.
- If a matching catch block is found, the code in that block is executed and the control moves to finally block.
- If no match is found the runtime system searches the call stack as discussed previously.
- The finally block is optional block that always executes after the try-catch block. It's used for releasing resources like closing files or database connections.

```
try {
   // Code that may throw an exception
} catch (ExceptionType e) {
   // Code to handle the exception
} finally {
   // cleanup code
}
```

```
try {       //Multiple Exceptions
   // Code that may throw an exception
} catch (ArithmeticException e) {
   // Code to handle the exception
} catch(ArrayIndexOutOfBoundsException e){
   //Code to handle the another exception
} catch(NumberFormatException e){
   //Code to handle the another exception
}
```

catch blocks for more specific exceptions must be placed before the catch blocks for more general exceptions to avoid a compile-time error.

Nested try-catch: you can place try-catch block inside another try block.

You can write multiple try-catch blocks one after another inside the same method.

(Each block is independent and handles a different piece of code)

---

throw: is used to explicitly throw an exception from a method or any block of code. We can throw either checked or un checked exception.

throw Instance

//Where instance is an object of type Throwable (or its subclasses)

throw new ArithmeticException("Cannot divide by zero");

بتتعمد ترمي استثناء بنفسك سواء جاهز أو أنت اللي عامله

---

throws: is used in the method signature to declare that this method might throw one of the listed exceptions.

```
public void readFile() throws IOException {
   // method code
}
returnType methodName(parameters) throws ExceptionType1, ExceptionType2,
ExceptionType3 {
   // Code that might throw any of the declared exceptions
}
```

We use the throws keyword to delegate the responsibility of exception handling to the caller (It may be a method or JVM) then the caller method has to handle that exception. (with try-catch)

We must handle the checked exception to prevent the compile time error by using throws.

throws بتعلن عن وجود Checked Exception داخل الميثود سواء جاهز أو أنا اللي عامله وبتهرب من التعامل معه.

```
class Geeks {

   static void fun() throws IllegalAccessException {
      System.out.println("Inside fun(). ");
```

```
        throw new IllegalAccessException("demo");
    }

    public static void main(String args[]) {
        try {
            fun();
        }
        catch (IllegalAccessException e) {
            System.out.println("Caught in main.");
        }
    }
}
```

User-Defined Exception = Custom Exception

Create your own exception class that extends Exception and throw that exception using the "throw" keyword.

Custom Exception can be:

- Checked Exception: it extends the Exception class.
- Unchecked Exception: it extends the Runtime Exception Class.
  (RuntimeException = UncheckedException)

```
public class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
//constructor to initialize the exception with custom message
    }
}
// Usage inside method or any block
try {
    throw new CustomException("This is a custom checked exception");
} catch (CustomException e) {
    System.out.println(e.getMessage());
}
```

```
// Custom Unchecked Exception
class DivideByZeroException extends RuntimeException {
    public DivideByZeroException(String m) {
        super(m);
    }
}
//Using the custom exception
public class Main {
    public static void divide(int a, int b) {
        if (b == 0) {
            throw new DivideByZeroException("Division by zero is not allowed.");
        }
        System.out.println("Result: " + (a / b));
    }
    public static void main(String[] args) {
        try {
            divide(10, 10);
        } catch (DivideByZeroException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
try {
    System.out.println(10 / 0);
}
catch (Exception e) { // Exception e = new ArithmeticException("/ by zero"); Upcasting
    // RuntimeException extends Exception
    System.out.println(e.getMessage());
}
```

Object Cloning - استنساخ الكائن

Creating an exact copy of an object using the clone() method

It creates a new instance of the class and initializes all its fields with exactly the contents of the corresponding fields of this object.

Human ahmed = new Human(25, "Ahmed Saif");

Human twin = new Human(ahmed);//Create new object (X Memory)

Marker Interface: Interface that doesn't contain any methods or constants (Empty Interface).

الميثود **clone()** الموجودة في كلاس Object لديها شرط داخلي: قبل أن تقوم بالنسخ فإنها تتحقق هل الكائن المطلوب نسخه الكلاس الخاص به implements Cloneable ؟

Method **clone()** exists inside Object class.

Shallow Copy – النسخ السطحي

```
public class Human implements Cloneable {
   int age;
   String name;
   int[] array;
   public Human(int age, String name) {
      this.age = age;
      this.name = name;
      this.array = new int[]{1,2,3,4,5};
   }
   @Override
   public Object clone() throws CloneNotSupportedException {
      return super.clone();
   }
}
public class Main {
   public static void main(String[] args) throws CloneNotSupportedException {
      Human ahmed = new Human(25, "Ahmed Saif");
      Human twin = (Human) ahmed.clone();
       /* Object object = new Human(25, "Ahmed Saif");
       return object
       Human twin = (Human) object; */
      System.out.println(twin.name);
      twin.array[0] = 100; // change in array affects object copy
      System.out.println(Arrays.toString(ahmed.array));
   }
```

```
}
```

In shallow copy changes to immutable fields like String do not affect each other.

Deep Copy

```
@Override // Replace clone() in previous Human class
   public Object clone() throws CloneNotSupportedException {
      Human twin =  (Human) super.clone(); // This is Shallow Copy
      twin.array = new int[twin.array.length];
      for (int i = 0; i < twin.array.length; i++) {
         twin.array[i] = this.array[i];
      }
      return twin;
   }

import java.util.Arrays;
public class Main {
   public static void main(String[] args) throws CloneNotSupportedException {
      Human human1 = new Human(25, "Ahmed Saif");
      Human human2 = (Human) human1.clone();
      human2.array[0] = 100;
      System.out.println(Arrays.toString(human2.array));
      System.out.println(Arrays.toString(human1.array));
   }
}
```

Data Structures

Main Operations: Add - Remove - Update - Select - Search

Array(Steps): (n + 1) – (n + 1) – (1) – (1) – (n)

(n + 1) steps = O(n)

Array: Static (Fixed Size) – Dynamic

Linked List(Steps): (n + 2) – (n + 2) – (n + 1) – (n + 1) – (n)

Big-O tells you the shape of the curve (Input Growth : Performance)

In (add / remove) Theoretical complexity is the same O(n) but linked list performs better (no shifting).

In (update / select) array performs better.

Stack | Queue (push / pop) مكدس | طابور

Stack: Last In First Out (LIFO)

Queue: First In First Out (FIFO)

Map: element (Key | Value) key and value can have different data types. Key is unique.

Stack:

- **stores method calls + local variables**
- **fast and organized (LIFO)**
- **memory is automatically freed when the method ends**

Heap:

- **stores objects (created with new)**
- **slower, but can hold more**
- **Garbage Collector cleans unused objects**

```
String name1 = "Ahmed";
String name2 = "Ahmed";
//The same object in String pool in heap
String name3 = new String("Ahmed");
//Even if "Ahmed" is in the pool, it creates a new object in the heap
```

لو عندك two objects بنفس المحتوى واستخدمت الهاش كود الافتراضية في كلاس أوبجكت هيتعملهم هاش كود مختلف وكل واحد هيتخزن في باكيت إنما لو عملت override لهاش كود هيطلع نفس الهاش كود لنفس المحتوى وفي الحالة دي بيضطر يستخدم equals عشان يقارنهم فلازم تعمل override للاتنين عشان تقارن المحتوى وترجع true فيمنع التكرار في الهاش ست وما شابه إنما لو رجعت false يبقى كأنه عنصر مختلف وهيتخزن في نفس الباكيت.

```
import java.util.Objects;
@Override
   public int hashCode() {
      return Objects.hash(name, id);
   }
```

أي كوليكشن فيها كلمة hash بتحتاج hashCode() عشان توصل لمكان العنصر بسرعة وتمنع التكرار.

Stream API

Introduced in Java 8 to process collections in a functional style. (java.util.stream)

Streams make it easier to perform operations such as filtering, mapping, reducing and collecting data without writing complex loops.

Works with Collection objects such as List, Set and Map.