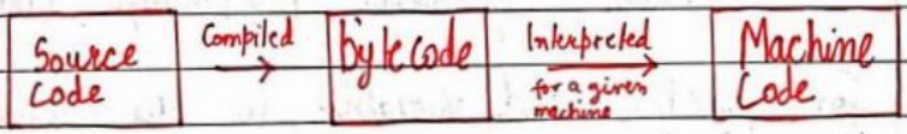
Java is an Object Oriented programming language developed by Sun Microsystems of USA in 1991

It was originally called Oak by James Goslin
↳ one of the inventors of Java!

JAVA = Purely Object Oriented

How JAVA Works?
Java is compiled into the bytecode and then it is interpreted to machine code



JAVA Installation
Go to Google & type "Install JDK" ⇒ Installs JAVA JDK
Go to Google & type "Install IntelliJ Idea" ⇒ Installs JAVA IDE

JDK → JAVA Development Kit = Collection of tools used for developing and running Java programs

JRE → JAVA Runtime Environment = Helps in executing programs developed in JAVA

URBAN EDGE

Basic Structure of a Java Program

```
package com.company;          → Groups classes!

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

Naming Conventions

→ For classes, we use PascalConvention. First and subsequent characters from a word are capital letters (uppercase)
Example:
Main, MyScanner, MyEmployee, CodeWithHarry

→ For functions and variables, we use camelCaseConvention. Here first character is lowercase and the subsequent characters are uppercase like below:
main, myScanner, myMarks, codeWithHarry

URBAN EDGE

Chapter 1 - Variables and datatypes

Just like we have some rules that we follow to speak english (the grammar), we have some rules to follow while writing a Java program. The set of these rules is called syntax.
↳ Vocabulary & Grammar of Java.

Variables

A variable is a container that stores a value. This value can be changed during the execution of the program.
Example:

```
int number = 8;
```

int → Data type; number → variable name; 8 → Value it stores!

Rules for declaring a variable name

We can choose a name while declaring a Java variable if the following rules are followed:

1. Must not begin with a digit → `int 1arry;` is invalid!
2. Name is case sensitive → harry and Harry are different!
3. Should not be a keyword (like Void)
4. White Space not allowed. → `int Code With Harry;` is invalid
5. Can contain alphabets, $ character, _ character and digits if the other conditions are met.

Data Types

Data types in Java fall under the following categories
1. Primitive Data Types (Intrinsic)
2. Non-Primitive Data Types (Derived)

URBAN EDGE

Primitive Data Types

Java is Statically typed. → variables must be declared before use!

There are 8 primitive data types supported by Java:

1) byte →
   - Value ranges from -128 to 127
   - Takes 1 byte
   - Default value is 0

2) Short →
   - Value ranges from $-(2^{16})/2$ to $(2^{16})/2 - 1$
   - Takes 2 bytes
   - Default value is 0

3) int →
   - Value ranges from $-(2^{32})/2$ to $(2^{32})/2 - 1$
   - Takes 4 bytes
   - Default value is 0

4) float →
   - Value ranges from (See Docs)
   - Takes 4 bytes
   - Default value is 0.0f

5) long →
   - Value ranges from $-(2^{64})/2$ to $(2^{64})/2 - 1$
   - Takes 8 bytes
   - Default value is 0

6) double →
   - Value ranges from (See docs)
   - Takes 8 bytes
   - Default value is 0.0d

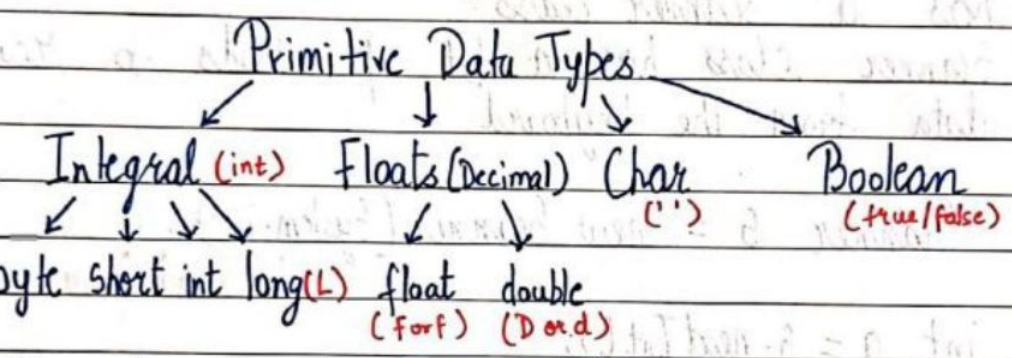
7) char →
   - Value ranges from 0 to 65535 $(2^{16}-1)$
   - Takes 2 bytes → because it supports unicode
   - Default value is '\u0000'

URBAN EDGE

8. boolean →
- Value can be true or false
- Size depends on JVM
- Default value is false

Quick Quiz : Write a Java program to add three numbers.

How to choose data types for our Variables



In order to choose the data type we first need to find the type of data we want to store. After that we need to analyze the Min & Max value we might use

Literals

A constant value which can be assigned to the variable is called as a literal

101 → Integer literal
10.1f → Float literal
10.1 → double Literal (default type for decimals)
'A' → character literal
true → boolean Literal
"Harry" → String Literal

URBAN EDGE

Keywords

Words which are reserved and used by the Java Compiler. They cannot be used as an Identifier.

↓↓

Go to docs.oracle.com for a comprehensive list!

Reading data from the Keyboard

In order to read data from the keyboard, Java has a Scanner class.

Scanner class has a lot of methods to read the data from the keyboard

```
Scanner s = new Scanner(System.in);
```

↳ Read from the keyboard

```
int a = s.nextInt();
```

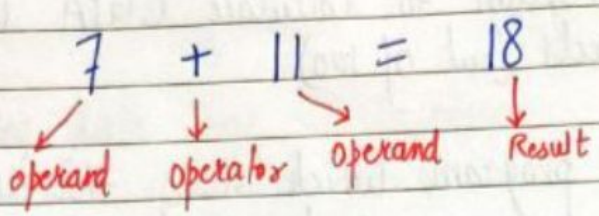
↳ Method to read from the keyboard (Integer in this case)

Exercise 1.1

Write a Program to Calculate percentage of a given student in CBSE board exam. His marks from 5 subjects must be taken as input from the keyboard. (Marks are out of 100).

Chapter 2 - operators and Expressions

Operators are used to perform operations on variables and values.



Types of operators

- → Arithmetic operators → +, -, *, /, %, ++, --
- → Assignment operators → =, +=
- → Comparison operators → ==, >=, <=
- → Logical operators → &&, ||, !
- → Bitwise Operators → &, | (operates bitwise)

Arithmetic operators cannot work with booleans
% operator can work on floats & doubles

Precedence of operators

The operators are applied and evaluated based on precedence. For example (+, -) has less precedence compared to (*, /). Hence * & / are evaluated first.
In case we like to change this order, we use parenthesis

Associativity

Associativity tells the direction of execution of operators
It can either be Left to Right or Right to left

- * / → L to R
- + - → L to R
- ++, = → R to L

URBAN EDGE

Quick Quiz: How will you write the following expressions in Java?

$\frac{x-y}{2}$, $\frac{b^2-4ac}{2a}$, v^2-u^2 , $a*b-d$

Resulting data type after arithmetic operation
following table summarizes the resulting data types after arithmetic operation on them

R = b + S → int
R = S + i → int
R = l + f → float
R = i + f → float
R = C + i → int
R = C + S → int
R = l + d → double
R = f + d → double

b → byte f → float
S → short d → double
i → integer c → character
l → long

Increment and Decrement operators
a++ , ++a → Increment operators → Data type
a-- , --a → Decrement operators → remains same

These will operate on all data types except booleans

Quick Quiz: Try increment and decrement operators on a Java variable

a++ → first use the value and then increment
++a → first increment the value then use it

Quick Quiz: What will be the value of the following expression (x).

```
int y = 7;
int x = ++y * 8;
```

Value of x?

```
Char a = 'B';
a++;
```
→ a is now 'C'

URBAN EDGE

Chapter 3 - Strings

A string is a sequence of characters
A string is instantiated as follows:

```
String name;
name = new String("Harry");
```

String is a class but can be used like a data type:

[Strings are immutable and cannot be changed.]

```
String name = "Harry";
```

name → Reference
"Harry" → Object

Different ways to print in Java

We can use the following ways to print in Java:

1. System.out.print() → No newline at the end!
2. System.out.println() → Prints a new line at the end
3. System.out.printf()
4. System.out.format()

```
System.out.printf("%c", ch)
```

- %d for int
- %f for float
- %c for char
- %s for string

String Methods

String methods operate on Java Strings. They can be used to find length of the string, convert to lowercase, etc.

Some of the commonly used String methods are :-

String name = "Harry";
0 1 2 3 4

1. name.length() → Returns length of String name. (5 in this case)

2. name.toLowerCase() → Returns a new String which has all the lowercase characters from the String name.

3. name.toUpperCase() → Returns a new String which has all the lowercase characters from the String name.

4. name.trim() → Returns a new String after removing all the leading and trailing spaces from the original String.

5. name.substring(int start) → Returns a substring from start to the end. substring(3) returns "ry"
[Note that index starts from 0]

6. name.substring(int start, int end) → Returns a substring from start index to the end index. Start index is included and end is excluded

7. name.replace('r', 'p') → Returns a new string after replacing r with p. Happy is returned in this case.
(char ↑ char ↑)

8. name.startsWith("Ha") → returns true if name starts with String "Ha". true in this case!
(String)

9. name.endsWith("ry") → returns true if name ends with string "ry". true in this case.
(String)

10. name.charAt(2) → returns character at a given index position. r in this case!
(int)

11. name.indexOf(s) → returns the index of the given String.
(str) For ex: name.indexOf("ar") returns 1 which is the first occurence of ar in String "Harry", -1 otherwise

12. name.indexOf("s", 3) → returns the index of the given String starting from the index 3 (int). -1 is returned in this case!

13. name.lastIndexOf("r") → returns the last index of the given String. 3 in this case!

14. name.lastIndexOf("r", 2) → returns the last index of the given string before index 2.

15. name.equals("Harry") → returns true if the given String is equal to "Harry" false otherwise [Case Sensitive]

URBAN EDGE

16 name.equalsIgnoreCase("harry") → returns true if two Strings are equal ignoring the case of characters.

Escape Sequence Characters
Sequence of characters after backslash '\'
= Escape sequence characters

Escape sequence characters consist of more than one characters but represents one character when used within the Strings.

Examples: \n, \t, \', \\, etc.
newline Tab single quote backslash

URBAN EDGE

Chapter 4 - Conditionals in Java

Sometimes we want to watch comedy videos on YouTube if the day is Sunday.
Sometimes, we order junk food if it is our friend's birthday in the hostel
You might want to buy an Umbrella if its raining and you have the money.
You order the meal if aloo or your favorite bhindi is listed on the menu.

All these are decisions which depends on a certain condition being met.
In Java, we can execute instructions on a condition being met.

Decision making Instructions in Java
→ If - Else Statement
→ Switch statement

If - else Statement
The syntax of an If - Else statement in C looks like that of C++ and JavaScript. Java has a similar syntax too. It looks like:

```
if (condition-to-be-checked) {
    Statements-if-condition-true;
}
else {
    Statements-if-condition-false;
}
```

URBAN EDGE

Code Example:

```
int a = 29;
if ( a > 18) {
    System.out.println(" You can drive");
}
```

Note that the else block is optional

Relational Operators in Java
Relational operators are used to evaluate conditions (true or false) inside the if statements.
Some examples of relational operators are:

==, >=, >, <, <=, !=

== : equals
>= : greater than or eq to
!= : Not equals

Note: '=' is used for assignment where as '==' is used for equality check.

The condition can be either true or false.

Logical Operators
&&, || and ! are most commonly used logical operators in Java
These are read as:

&& → AND
|| → OR
! → NOT

⇒ Used to provide logic to our JAVA programs

URBAN EDGE

AND operator

Evaluates to true if both the conditions are true

Y && Y = Y
Y && N = N
N && Y = N
N && N = N

Y → true
N → false

OR Operator

Evaluates to true when at least one of the conditions is true.

Y || Y = Y
Y || N = Y
N || Y = Y
N || N = N

Y → true
N → false

NOT Operator

Negates the given logic (true becomes false and false becomes true)

! Y = N
! N = Y

Y → true
N → false

else if clause

Instead of using multiple if statements, we can also use else if along with if thus forming an if-else-if-else ladder

Using such kind of logic reduces indents. last else is executed only if all the conditions fail.

```
if (condition) {
    // Statements;
}
else if {
    // Statements;
}
else {
    // Statements;
}
```

Switch Case Control Instruction

Switch - Case is used when we have to make a choice between number of alternatives for a given variable

```
Switch (var) {
    Case C1 :
        // code;
        break;
    Case C2 :
        // Code
        break;

    Case C3 :
        // Code
        break
    default :
        // Code

}
```

URBAN EDGE

Var can be an integer, character or string in Java.

A switch can occur within another but in practice this is rarely done

URBAN EDGE

Chapter 5 - Loop Control Instruction

Sometimes we want our programs to execute a few set of instructions over and over again for example - print 1 to 1000, print multiplication table of 7, etc.
Loops make it easy for us to tell the Computer that a given set of instructions need to be executed repeatedly.

Types of Loops

Primarily, there are three types of loops in Java:

1. While loop
2. do-while loop
3. for loop

We will look into these one by one.

While loops

```
While (boolean condition)
{
    // Statement
}
```

→ This keeps executing as long as the condition is true.

If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

URBAN EDGE

Quick Quiz: Write a program to print natural numbers from 100 to 200.

do-while loop

This loop is similar to a while loop except the fact that it is guaranteed to execute at least once.

```
do {
  // code
} while(condition);        → Note this Semicolon
```

while → checks the condition & executes the code
do-while → Executes the code & then checks the condition

Quick Quiz: Write a program to print first n natural numbers using do-while loop.

For Loop

The syntax of a for loop looks like this:

```
for (initialize; check_bool_expression; update) {
  // code;
  }
```

A for loop is usually used to execute a piece of code for specific number of times.

Quick Quiz: Write a program to print first n odd numbers using a for loop.

URBAN EDGE

Decrementing for loop

```
for(i=7; i!=0; i--){
    System.out.println(i);
}
```

This for loop keeps running until i becomes 0.

Quick Quiz: Write a program to print first n natural numbers in reverse order

break statement.
The break statement is used to exit the loop irrespective of whether the condition is true or false.
Whenever a "break" is encountered inside the loop, the control is sent outside the loop.

Continue statement
The continue statement is used to immidiately move to the next iteration of the loop.
The control is taken to the next iteration thus skipping everything below "continue" inside the loop for that iteration.

In a Nut shell...
1. break statement completely exits the loop
2. continue statement skips the particular iteration of the loop.

URBAN EDGE

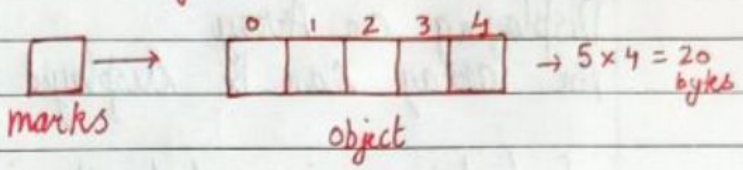
Chapter 6 - Arrays

Array is a collection of similar types of data

Use Case : Storing marks of 5 students

```
int [] marks = new int [5]
```
⇒ [dataType ArrName;]

reference — object



Accessing Array Elements

Array elements can be accessed as follows

```
marks[0] = 100
marks[1] = 70
  :
  :
marks[4] = 98
```

⇒ Note that index starts from 0

So in a nut shell, this is how array works:

1. `int [] marks;` → Declaration!
   `marks = new int[5];` → Memory Allocation!
2. `int [] marks = new int[5];` → Declaration + Memory Allocation!
3. `int [] marks = {100, 70, 80, 71, 98}` → Declare + Initialize!

Array indices starts from 0 and goes till (n-1) where n is the size of the array.

URBAN EDGE

Array length

Arrays have a length property which gives the length of the array

marks.length → gives 5 if marks is a reference to array with 5 elements

Displaying an Array

An array can be displayed using a for loop:

```
for(int i=0; i<marks.length; i++)
{
    Sout(marks[i]);
}
```

⇒ Array Traversal

Quick Quiz: Write a Java program to print the elements of an array in reverse order.

For-each loop in Java

Array elements can also be traversed as follows:

```
for(int element : Arr){
    Sout(element);
}
```

⇒ Prints all the elements

Multidimensional Arrays

Multidimensional Arrays are Array of Arrays
Each element of a M-D array is an array itself
marks in the previous example was a 1-D array.

URBAN EDGE

Multidimensional 2-D Array

A 2-D array can be created as follows:

```
int [][] flats = new int [2][3]
```

↳ A 2-D array of 2 rows + 3 Columns

We can add elements to this array as follows

```
flats [0][0] = 100
flats [0][1] = 101
flats [0][2] = 102
   :
```

& so on!

This 2-D array can be visualised as follows:

	[0] Col 1	[1] Col 2	[2] Col 3
[0] Row 1	(0,0)	(0,1)	(0,2)
[1] Row 2	(1,0)	(1,1)	(1,2)

Similarly a 3-D array can be created as follows:

```
String [1][][] arr = new String [2][3][4]
```

URBAN EDGE

Chapter 7 - Methods in Java

Sometimes our program grows in size and we want to separate the logic of main method to other methods
For instance - If we are calculating average of a number pair 5 times, we can use methods to avoid repeating the logic.
↳ DRY = Dont Repeat yourself.

Syntax of a Method

A method is a function written inside a class. Since Java is an Object oriented language, we need to write the method inside some class

```
dataType name () {
  // Method body
  }
```

Following method returns sum of two numbers

```
  ↳ Return type
int mySum( int a, int b) {
   int c = a+b;
   return c;                    → Return value
   }
```

Calling a Method

A method can be called by creating an object of the class in which the method exists followed by the method call:

```
Calc obj = new Calc(); → Object Creation
obj.mySum(a, b);       → Method call upon an object
```

The values from the method call (a and b) are copied to the a and b of the function mySum. Thus even if we modify the values a and b inside the method, the values in the main method will not change.

Void return type
When we dont want our method to return anything, we use void as the return type

Static keyword
Static keyword is used to associate a method of a given class with the class rather than the object. Static method in a class is shared by all the objects

Process of method invocation in Java
Consider the method Sum:

```
int Sum (int a, int b)
    {
    return a+b;
    }
```

The method is called like this:

```
Calc obj = new Calc();
c = obj.Sum( 2, 3)
```

The values 2 and 3 are copied to a and b and then a+b = 2+3 = 5 is returned in c which is an integer.

Note: In case of Arrays, the reference is passed. Same is the case for Object passing to methods.

URBAN EDGE

Method Overloading

Two or more methods can have same name but different parameters. Such methods are called Overloaded methods.

```
Void foo()
Void foo(int a)          => Overloaded function foo
int foo(int a, int b)
```

Method overloading cannot be performed by changing the return type of methods

Variable Arguments (Varargs)

A function with vararg can be created in Java using the following syntax:

```
public static void foo(int ... arr)
{
  // arr is available here as int[] arr
}
```

foo can be called with Zero or more arguments like this:

foo(7) foo(7,8,9) foo(1,2,7,8,9)

We can also create a function bar like this

```
public static void bar(int a, int arr)
{
  // code
}
```

↳ Atleast one integer is required now

bar can be called as bar(1), bar(1,2), bar(1,7,9,11) etc.

URBAN EDGE

Recursion

A function in Java can call itself. Such calling of function by itself is called recursion.

Example: Factorial of a number

$$factorial(n) = n * factorial(n-1) \quad \forall\ n \geq 1$$

Quick Quiz: Write a program to calculate (recursion must be used) factorial of a number in Java ?

URBAN EDGE

Chapter - 8 : Introduction to OOPs

Object Oriented programming tries to map code instructions with real world making the code short and easier to understand

What is Object Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object Oriented Programming.

What is DRY?

DRY stands for - Do not repeat yourself

↳ Focuses on code reusability

Class

A class is a blueprint for creating objects.

JEE Application form ⇒ Filled by an Student ⇒ Application for that Student

Class ⇒ Object Instantiation ⇒ Object

Contains info to create a valid object.

Object

An Object is an instantiation of a class. When a class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

URBAN EDGE

How to model a problem in OOPs
We identify the following :

Noun → Class → Employee
Adjective → Attributes → name, age, salary
Verb → Methods → getSalary(), increment()

OOPs Terminology

1. Abstraction → Hiding internal details [show only essential info.!]

⇒ use this phone without bothering about how it was made

2. Encapsulation → The act of putting various components together (in a capsule).

⇒ Laptop is a single entity with Wifi + Speaker + Storage in a single box!

In Java, encapsulation simply means that the sensitive data can be hidden from the users

3. Inheritance → The act of deriving new things from existing things.

Rickshaw ⇒ E-Rickshaw
Phone ⇒ Smart Phone

Implements DRY!

4. Polymorphism → One entity many forms

Smartphone → Phone Smartphone → Calculator

URBAN EDGE

Writing a Custom Class

We can write a custom class as follows:

```
public class Employee {
    int id;          → Attribute 1
    String name;     → Attribute 2
}
```

Any real world object = Properties + Behaviour
Object in OOPs = Attributes + Methods.

A class with Methods

We can add methods to our class Employee as follows:

```
public class Employee {
    public int id;
    public String name;

    public int getSalary() {
        // code
    }
    public void getDetails() {
        // code
    }
};
```

URBAN EDGE

Chapter 9 - Access Modifiers & Constructors

Access Modifiers

Specifier where a property / method is accessible
There are four types of access modifiers in Java:

1. Private
2. Default
3. Protected
4. Public

Getters and Setters

Getter → Returns the value [accessors]
Setter → Sets/Updates the value [mutators]

Example :

```
public class Employee {
    private int id;
    private String name;

    public String getName() {
        return name;
        }
    public void setName() {
        this.name = "Your-name";
        }
    public void setName(String n) {
        this.name = n;
        }
}
```

URBAN EDGE

Quick Quiz : Use these getters and setters from the main method.

Constructors in Java
A member function used to initialize an object while creating it.

```
Employee harry = new Employee();
harry.setName("Harry Bhai");
```

In order to write our own constructor, we define a method with name same as class name.

```
public Employee(){
    name = "Your Name");
    }
```

Constructor Overloading in Java
Constructors can be overloaded just like other methods in Java. We can overload the Employee Constructor like below:

```
public Employee(String n) {
    name = n;
    }
```

Note : ① Constructors can take parameters without being overloaded.

② There can be more than two overloaded Constructors

URBAN EDGE

Chapter 10 - Inheritance

Inheritance is used to borrow properties & methods from an existing class

Phone → SmartPhone

SuperClass → SubClass SubClass extends SuperClass

Declaring Inheritance in Java

Inheritance in Java is declared using extends keyword

Superclass
↓
Subclass ⇒ Subclass extends the Superclass

More Examples

Vehicle	Animal	Animal	Vehicle
↓	↓	↓	↓
Car	Dog	Cat	Truck

When a class inherits from a superclass, it inherits parts of superclass methods and fields.
Java doesn't support multiple inheritance ie two classes cannot be super classes for a subclass.

Code Example

Inheritance in Java is declared using extends keyword

```
public class Dog extends Animal {→ Inheriting Dog from Animal Class!!
    // Code
}
```

Quick Quiz: Create a class Animal and Derive another class Dog from it.

Constructors in Inheritance

When a Derived class is extended from the Base class, the Constructor of the Base class is executed first followed by the Constructor of the derived class.

For the following Inheritance hierarchy, the constructors are executed in the order ① → ② → ③

C_1 → Parent
↓
C_2 → child
↓
C_3 → Grand child

↓ Constructors execute in top to bottom order!

Constructors during Constructor Overloading

When there are multiple constructors in the parent class, the constructor without any parameters is called from the child class.

If we want to call the constructor with parameters from the parent class, we can use Super keyword

```
Super ( a , b );
```
→ Calls the constructor from the parent class which takes 2 Variables

this Keyword

this is a way for us to reference an object of the class which is being created/referenced.

```
this.area = 2
```
→ this is a reference to current object

Super Keyword

A reference variable used to refer immediate parent class object

→ Can be used to refer immediate parent class instance variable
→ Can be used to invoke parent class methods.
→ Can be used to invoke parent class constructors.

Method Overriding

If the child class implements the same method present in the parent class again, it is known as method overriding

URBAN EDGE

Chapter 11 - Abstract Classes & Interfaces

What does Abstract (class) mean ?
Abstract in english means → existing in thought or as an idea without concrete existance

Abstract method
A method that is declared without an implementation

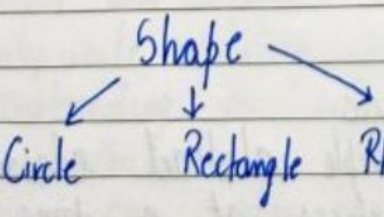
```
abstract void moveTo (double x, double y)
```

Abstract Class
If a class includes abstract methods, then the class itself must be declared abstract, as in :

```
public abstract class PhoneModel {
    abstract void switchoff();
    // more code
    }
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the methods in parent class. If it doesnt, it must be declared abstract

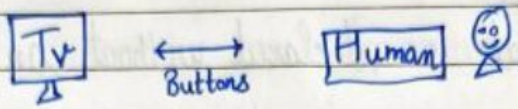
An Example



Note - It is possible to create reference of an abstract class
It is not possible to create an object of an abstract class

We can also assign reference of an abstract class to the object of a concrete subclass.

Interfaces in Java
Interface in english is a point where two systems meet and interact



In Java interface is a group of related methods with empty bodies

An Example

```
interface Bicycle {
    void applyBrake(int decrement);
    void speedUp(int increment);
    }

class AvonCycle implements Bicycle {
    int speed = 7;
    void applyBrake(int decrement) {
        speed = speed - decrement;
        }
    void speedUp(int increment) {
        speed = speed + increment;
        }
```

Abstract class vs Interfaces
We cant extend multiple abstract classes but we can implement multiple interfaces at a time.
Interfaces are meant for dynamic method dispatch

and run time polymorphism

Is multiple inheritance allowed in Java?
Multiple inheritance face problems when there exist methods with same signature in both the super classes.
Due to such problems, Java does not support multiple inheritance directly but the similar concept can be acheived using Interfaces
A class can implement multiple Interfaces and extend a class at the same time.

Note: ① Interfaces in Java is a bit like the Class but with a significant difference.
② An Interface can only have method signatures, Constant ← fields and default methods.
③ The class implementing an Interface needs to on declare the methods (not fields)
④ You can create a reference of Interfaces but not the Object
⑤ Interface methods are public by default.

Default methods
An interface can have static and default methods.
Default methods enable us to add new functionality to existing Interfaces.
This feature was introduced in Java 8 to ensure backward compatibity while updating an Interface.
Classes implementing the interface need not implement the default methods.
Interfaces can also include private methods for default methods to use.

URBAN EDGE

Inheritance in Interfaces

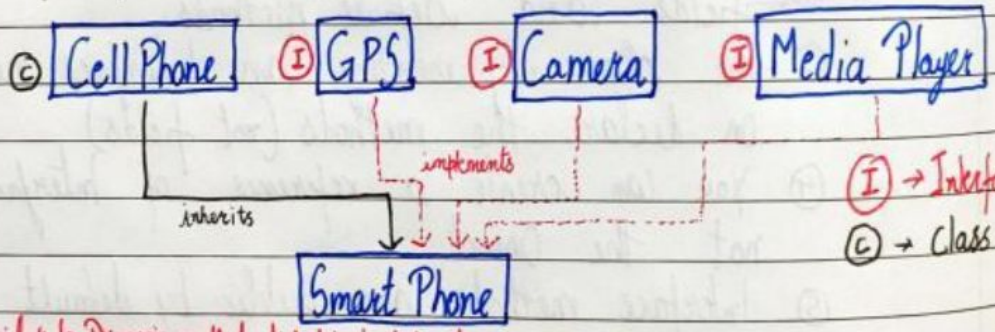
Interfaces can extend another interfaces:

```
public interface Interface1 {
        Void meth1();
        }

public interface Interface2 extends Interface1 {
        void meth2();
        }
```

Remember that interface cannot implement another interface, only classes can do that!

Polymorphism using Interfaces



Similar to Dynamic method dispatch in Inheritance

```
GPS g = new SmartPhone(); → Can only use GPS methods
SmartPhone s = new SmartPhone(); → Can only use SmartPhone methods
```

Implementing an Interface forces method implementation.

URBAN EDGE

Chapter 12 - Packages

Interpreter Vs Compiler

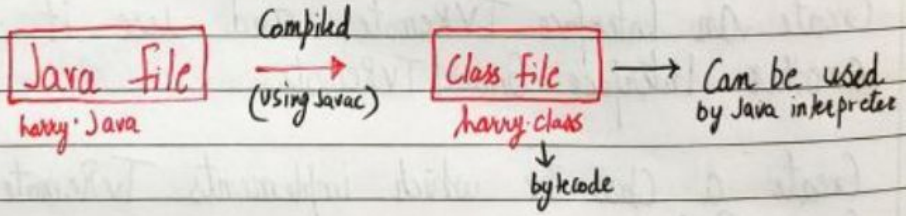
Interpreter translates one statement at a time into machine code.

Compiler scans the entire program and translates whole of it into machine code.

Interpreter	Compiler
* One statement at a time	* Entire program at a time
* Interpreter is needed everytime	* Once compiled it is not needed
* Partial execution if error	* No execution if an error occurs
* Easy for programmers	* Usually not as easy as Interpreted ones

Is Java Compiled or Interpreted?

Java is a hybrid language → both compiled as well as interpreted



- → A JVM can be used to Interprete this bytecode
- → This bytecode can be taken to any platform (Win/Mac/Linux) for execution
- → Hence Java is platform independent (write once run everywhere)

URBAN EDGE

Executing a Java Program

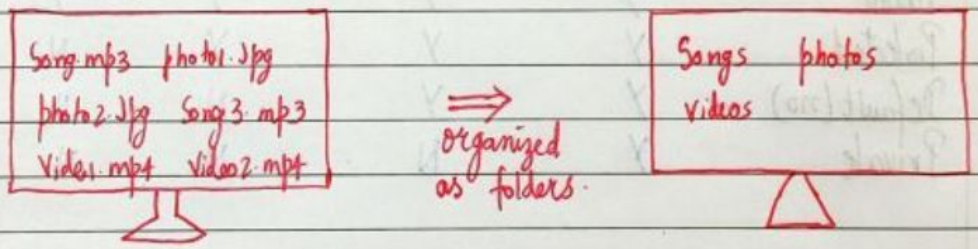
Javac Harry.java → Compiled
Java Harry.class → Interpreted

So fare the execution of our program was being managed by intelly Idea.
We can download a source code editor like VS Code to compile & execute our Java programs.

Packages in Java

A package is used to group related classes.
Packages help in avoiding name conflicts.
There are two types of packages:

* Built in packages → Java API
* User defined packages → Custom packages



1.class this.java my.mp3 ⟹ organized as packages
Song.java harry.java

Using a Java package

import java.lang.* → import everything from java.lang
import java.lang.String → import String from java.lang
s = new Java.lang.String("Harry") → Use without importing

Creating a package

Javac Harry.java → Creates Harry.class

javac -d . Harry.java → creates a package folder

↳ We can keep adding classes to a package like this

We can also create inner packages by adding "package.inner" as package name

(package = folder, inner = subfolder)

These packages once created can be used by other classes.

Access Modifiers in Java

Access modifiers determine whether other classes can use a particular field or invoke a particular method

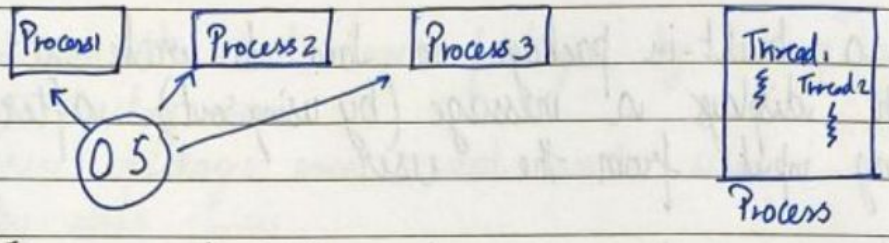
Can be public, private, protected or default (no modifier)

Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Default (no)	Y	Y	N	N
Private	Y	N	N	N

URBAN EDGE

Chapter 13 - Multithreading

Multiprocessing and multithreading both are used to acheive multitasking



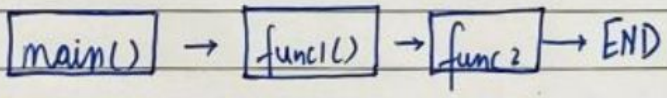
In a nut Shell ...

→ Threads use shared memory area.
→ Threads ⇒ Faster Context Switching
→ A Thread is light-weight whereas a process is heavyweight

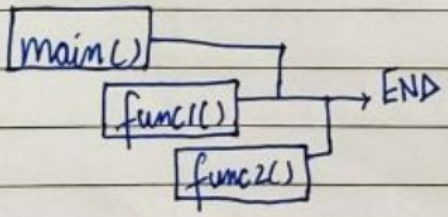
For Example → A word processor can have one thread running in foreground as an editor and another in the background auto saving the document!

Flow of Control in Java

1. Without threading:



2. With threading:



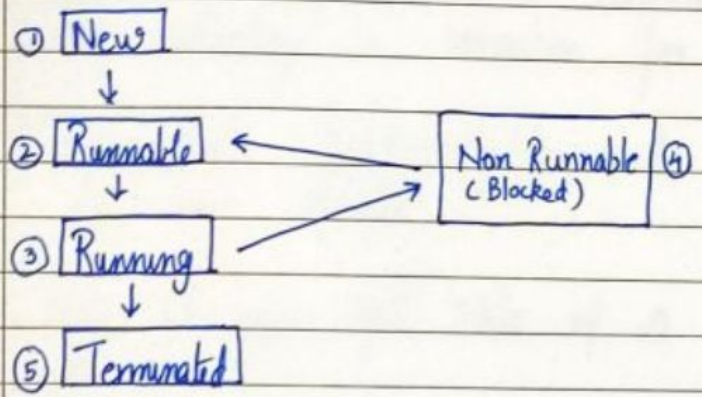
URBAN EDGE

Creating a Thread

There are two ways to create a thread in Java.

1. By extending Thread class
2. By implementing Runnable interface

Life cycle of a Thread



(1) New → Instance of thread created which is not yet started by invoking start()

(2) Runnable → After invocation of start() & before it is selected to be run by the scheduler.

(3) Running → After thread scheduler has selected it.

(4) Non Runnable → Thread alive, not eligible to run.

(5) Terminated → run() method has exitted

The Thread class

Below are the commonly used Constructors of Thread class:

(1) Thread()
(2) Thread(String name)
(3) Thread(Runnable r)
(4) Thread(Runnable r, String name)

URBAN EDGE

Methods of Thread class

Thread class offers a lot of methods such as run(), start(), join(), getPriority(), setPriority() etc.

More can be found on visiting Java docs

URBAN EDGE

Chapter 14 - Errors & Exceptions

No matter how smart we are, errors are our constant companions. With practice, we keep getting better at finding & correcting them.

There are three types of errors in Java.

1) Syntax errors
2) Logical errors
3) Runtime errors → Also called Exceptions!

Syntax Errors

When compiler finds something wrong with our program, it throws a syntax error.

```
int a = 9          → No semicolon, syntax error!
a = a+3;

d = 4;  → Variable not declared, Syntax error!
```

Logical errors

A logical error or a bug occurs when a program compiles and runs but does the wrong thing.

→ message delivered wrongly
→ Wrong time of chats being displayed
→ incorrect redirects!

Runtime Errors

Java may sometimes encounter an error while the program is running. These are also called exceptions!

URBAN EDGE

These are encountered due to circumstances like bad input and (or) resource constraints.
Ex: User supplies '5'+8 to a program which adds 2 numbers.

Syntax errors and logical errors are encountered by the programmer where as Runtime errors are encountered by the users.

Exceptions in Java
An Exception is an event that occurs when a program is executed disrupting the normal flow of instructions.
There are mainly two types of exceptions in Java:
1) Checked Exception → Compile time exceptions (Handled by Compiler)
2) Unchecked Exception → Runtime exceptions

Commonly Occuring Exceptions
Following are few commonly occuring exceptions in Java:
1) Null Pointer Exception
2) Arithmetic Exception
3) Array Index Out of Bound Exception
4) Illegal Argument Exception
5) Number format Exception

try-catch block in Java
In Java, exceptions are managed using try-catch blocks
Syntax:

```
try {
    // code to try }
catch (Exception e) {
    // Code if exception
    }
```

URBAN EDGE

Handling specific Exceptions

In Java, we can handle specific exceptions by typing multiple catch blocks.

```
try {
    // Code
    }
Catch (IOException e) {        → Handles all Exceptions of type IOException
    // Code
    }
Catch (ArithmeticException e) {  → Handles all Exceptions of type ArithmeticException
    // Code
    }
catch ( Exception e) {         → Handles all other Exceptions
    // Code
    }
```

Nested try-catch

We can nest multiple try-catch blocks as follows:

```
try {
    try {
        }
    Catch (Ex e){
        }
    }
Catch (Ex e){                  ⇒ Nested try-catch blocks
        }
```

Similarly, we can further nest try catch blocks inside the nested try catch blocks.

URBAN EDGE

Quick Quiz: Write a Java program that allows you to keep accessing an array until a valid index is given by the user.

Exception class in Java

We can write our custom Exceptions using Exception class in Java.

```
public class MyException extends Exception {
        //overridden methods
        }
```

The Exception class has following important methods:

1. String toString() → executed when sout(e) is ran
2. Void printStackTrace() → prints stack trace
3. String getMessage() → prints the Exception message

The Throw Keyword

The throw keyword is used to throw an exception explicitly by the programmer

```
if (b==0) {
    throw new ArithmeticException("Div by 0");
    }
else {
 return a/b;
    }
```

In a similar manner, we can throw user defined exceptions:

```
throw new MyException("Exception thrown");
```

URBAN EDGE

The throws exception

The Java throws keyword is used to declare an exception. This gives an information to the programmer that there might be an exception so, it is better to be prepared with a try catch block!

```
public void calculate (int a, int b) throws IOException {
        // code
    }
```

Java finally block

finally block contains the code which is always executed whether the exception is handled or not.

It is used to execute code containing instructions to release the system resources, close a connection etc.

URBAN EDGE

Advanced Java - 1

Collections Framework

A collection represents a group of object.
Java Collections provide Classes and Interfaces for us to be able to write code quickly and efficiently

Why do we need Collections

We need Collections for efficient storage and better manipulation of data in Java.
For ex: We use arrays to store integers but what if we want to

→ Resize this array?
→ Insert an element in between?
→ Delete an element in Array?
→ Apply certain operations to change this array?

How are Collections available

Collections in Java are available as Classes and Interfaces. Following are few commonly used Collections in Java:

* ArrayList → For variable size Collection
* Set → For distinct collection
* Stack → A LIFO data structure
* HashMap → For storing key-value pairs

Collection class is available in java.util package
Collection class also provides static methods for sorting, searching etc.

URBAN EDGE

Date & Time in Java

Java.time → package for Date & time in Java
↳ from Java 8 onwards

Before java 8, java.util package used to hold the date and time classes. Now these classes are deprecated

How java stores a Date?

Date in Java is stored in the form of a long numer. This long number holds the number of milliseconds passed since 1 Jan 1970

Java assumes that 1900 is the start year which means it calculates years passed since 1900 whenever we ask it for years passed.

System.currentTimeMillis() returns no of seconds passed. Once no of ms are calculated, we can calculate minutes, seconds & years passed

Quick Quiz: Is it save to store the no of ms in a variable of type long?

The Date class in Java

```
Date d = new Date();
System.out.println(d);
```

We can also use constructors provided by the Date class

URBAN EDGE

Java Date class has few methods which can be used. for ex: getDate(), getDay() etc.

All these methods are deprecated

Calendar Class in Java
Calendar is an abstract class that provides calendar related methods in Java

Calendar.getInstance → returns a Calendar instance based on current time

```
Calendar a = Calendar.getInstance();
a.getTime() → prints time
```

Calendar class methods

1. get method is used to get year, date, min, second

```
a.get(Calendar.SECOND)
a.get(Calendar.MINUTE)
a.get(Calendar.DATE)
a.get(Calendar.YEAR)
```

2. getTime method returns a Date object

3. Other methods can be looked up from the Java docs!

Gregorian Calendar Class
This class is used to create an instance of gregorian calendar
We can change the year month & date using Set method!

URBAN EDGE

Time Zone

TimeZone class is used to create Time Zones in Java. Some of the important methods of Time Zone class are:

getAvailableIDs() → get all the available IDs supported
getDefault() → get the default timezone
getID() → get the ID of a TimeZone

Java.time package

→ Available from Java 8 onwards
→ Capable of storing even nano seconds

Following as some of the most commonly used classes from java.time package.

LocalDate → Represents a Date
LocalTime → Represents a Time
LocalDateTime → Represents a Date + Time
DateTimeFormatter → Formatter for displaying & parsing date-time objects

Ultimate Java Quick Reference

1. // [comment]

Single line comment.

2. /* [comment] */

Multi line comment.

3. public

This can be imported publically.

4. import [object].*

Imports everything in object.

5. static

Going to be shared by every [object].

6. final

Cannot be changed; common to be defined with all uppercase.

7. double

Integer with numbers that can have decimals.

8. ;

Put after every command.

9. String

Just a string of characters.

10. Private

Can only be changed by a method.

11. int

Can store numbers from 2^-31 to 2^31.

12. fields are attributes

13. boolean

Can have true or false as the value.

14. { }

These are used to start and end a function, class, etc.

15. byte

These can store from -127 - 128.

16. long

Can store numbers from 2^127 to 2^-127.

17. char

Just lets you put in one chracter.

18. double

64-bit number with decimals.

19. float

32-bit number with decimals.

20. protected

Can only be accessed by other code in the package.

21. Scanner

This lets you get user input.

22. new [object constructor]

This will let you create a new object.

23. System.in

This lets you get data from the keyboard.

24. public [class]()

This will be the constructor, you use it to create new objects.

25. super()

This will create the superclass (the class it's inheriting).

Ultimate Java Quick Reference - CodeWithHarry

26. extends [class]

Makes the object a subclass of [object], [object] must be a superclass.

27. ++

Will increment the amount.

28. --

Will decrement the amount.

29. += [amount]

Increment by [amount]

30. -= [amount]

Decrement by [amount]

31. *= [amount]

Multiply by [amount]

32. /= [amount]

Divide by [amount]

33. System.out.println([text])

Will print something to the output console.

34. +

Can be used for concatenation. (ex. "6" + [var_here])

35. public static void main(String[] args)

This is your main function and your project will start in here.

36. System.out.print([text])

This prints stuff but there is no line break. (/n)

37. \n

Called a line break; will print a new line.

38. \t

This will print a tab.

39. if ([condition])

This will make it so if [condition] is true then it'll keep going.

40. &&

This means and.

41. !

This means not.

42. ||

This means or.

43. ==

This means equal to.

44. <

This means less than.

45. >

This means greater than.

46. >=

This means greater than or equal to.

47. [inputVarHere].hasNextLine()

This will return if there is a next line in the input.

48. this

Refer to the class that you are in.

49. [caller].next[datatype]()

This will get the [datatype] that you somehow inputted.

50. Create getters and setters

This will create the get methods and set methods for every checked variable.

51. [caller].hasNext[datatype]()

Ultimate Java Quick Reference - CodeWithHarry

This will return if it has the correct datatype within the input.

52. overloading

If you have different parameters you can call them whatever way you want.

53. parameters

These are the inputs of your function.

54. ([datatype])[variable]

This will convert [variable] into [datatype]. Also known as casting.

55. Math.random()

Generate an extremely percise string of numbers between 0 and 1.

56. Primitives

Just the basic data types which are not objects.

57. [x].toString()

Will convert [x] into a string.

58. [number].parse[numbertype]([string])

This will parse [number] into the [numbertype] with [string].

59. ^

Return true if there is one true and one false.

60. !=

Not equal too. (NEQ)

61. ([condition]) ? [amount] : [var]

This will be like a shortcut way to an if statement.

62. switch([variable])

This will do stuff with specific cases. (e.g. switch(hi){ case 2: (do stuff)})

63. case [value]:

This will do stuff if the case is the case.

64. break

Put that when you want to leave the loop/switch; should be at end of case.

65. default [value]:

This will do stuff if none of the cases in the switch statement was made.

66. for ([number]; [condition]; [operation])

This will start at [number] and then do [operation] until [condition] is met.

67. continue

This will just go back to the enclosing loop before reaching other code.

68. while ([condition])

This will basically do something while [condition] is true.

69. void

This means no return type.

70. return

This will return something when you call it to where it was called from .

71. do { } while ([condition])

Guarantees it will execute once even if [condition] isn't met.

72. printf("%[type] stuff here bah bla", [variable here])

This will let you use [variable here] with %s being where.

Ultimate Java Quick Reference - CodeWithHarry

73. System.out.printf([text])

Another way to print? // didn't quite get but ok then

74. [type] [returntype] [name]([parameters]) {

This is a way to create a method.

75. [type][[indexes]]

This will create an array with [indexes] amount of indexes; default infinite.

76. int[] something = new int[20];

This will just make an array of ints with 20 ints in it.

77. for ([object] [nameOfObject] : [arrayOfObject]) {

This will iterate through all of the arrayOfObject with object in use incrementing by 1 until done.

78. [object][[1]][[2]][[3]] [name] = {[value] [value] [value] \n [value] [value] [value]}

[1] is how many down in array, [2] how many accross in array, [3] how many groups

79. .length

This will get how long something is, text, amount of indexes in array, etc.

80. Arrays.copyOf([array], indexes);

This will copy the array and how many indexes into another array.

81. Arrays.toString([array])

Convert the whole array into one huge string.

82. Arrays.binarySearch([array], [object])

This will search for [object] in [array].