Cheat-Sheet JAVA

OOPs Basic concepts

INTERPRETER VS COMPILER

- **Compilers** & **Interpreters** are used to convert a program written in a high-level language into machine code understood by computers
- Java source is compiled to **Platform Independent Byte Code** (.class files). For execution, byte codes are interpreted directly by the Java Virtual Machine (JVM)

COMPILER C/C++,C# | Java (Hybrid)

- Scans the entire program and translates it as a whole into machine code & the errors (if any) are shown at the end together
- Compiled code runs faster
- It converts the source code into object code. So doesn't require source code for later execution.

INTERPRETER Python,Perl

- It scans program one statement at a time & errors are shown line by line.
- Interpreted code runs slower
- It does not convert source code into object code instead it scans it line by line. So requires source code for later execution.

POLYMORPHISM

more than one form

- The same entity (method or operator or object) can perform different operations in different scenarios.
- Example,
  Smartphone -> camera
  Smartphone -> calculator

COMPILE-TIME OR STATIC POLYMORPHISM

Method Overloading

- This allows us to have more than one method having the same name, if the parameters of methods are different
- overloading a static method in java, is the example of compile time polymorphism

RUN-TIME POLYMORPHISM

Method Overriding

- Declaring a method in sub class which is already present in parent class is known as method overriding
- method overriding is a run-time polymorphism

Dynamic Method Dispatch

- Dynamic method dispatch is the mechanism in which a call to an overridden method is resolved at run time instead of compile time.

Upcasting

- Phone p = new Smartphone();
- Upcasting allows only overridden methods to use

ABSTRACTION

idea but not having a concrete existence

- Abstraction is a process of hiding the implementation details and showing only functionality to the user
- A car is viewed as a car rather than its individual components
- achieved using Abstract classes and interfaces

ABSTRACT METHOD

- A method which is declared as abstract and does not have implementation is known as an abstract method.
- Example, abstract void printStatus();

ABSTRACT CLASS extends

- can have abstract and non-abstract methods
- It cannot be instantiated But can be referenced/subclass
- It can have constructors and static methods also
- It can have final methods which will force the subclass not to change the body of the method.

INTERFACE implements

- The interface can have only abstract methods without implementation **but** java 8 allows **default** methods - method with body which can be override
- Abstract methods must be public & NOT static
- Implementation of abstract class should be public
- Instance variables created in interface are always final but can be override
- Multiple inheritance
  Mobile = CellPhone class + Camera interface + TouchScreen interface + GPS interface + ...
  Polymorphism : GPS p = new Smartphone();

ACCESS MODIFIER

PUBLIC

- can be accessed from anywhere, just need to import

PRIVATE

- can be access only within the class

DEFAULT

- can be access only within the package itś default : no need to import

PROTECTED

- can be access within the package & outside the package through child class (inheritance)

INHERITANCE

Act of deriving new things from existing things

- smartphone from phone

ENCAPSULATION

Keeping various components together

- In java, Encapsulation simply means keeping sensitive data hidden from the users
- Getters & Setters

SOME BASIC COMMANDS

Creating Packages

```
$ Javac -d . *.java
```