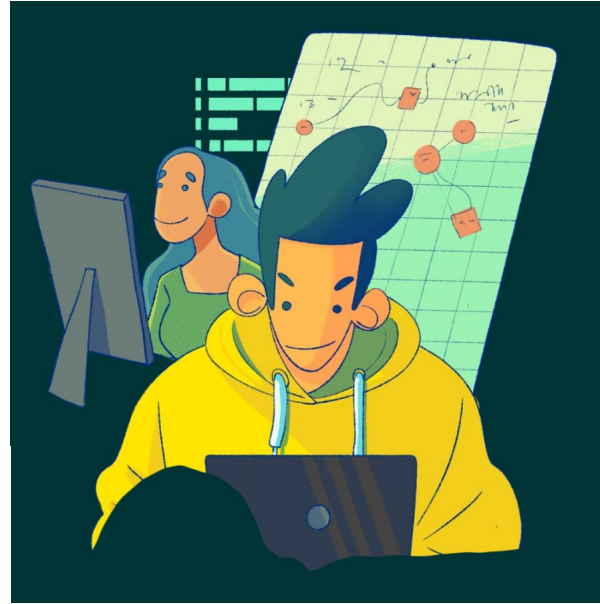


while (folksAreJoining() == true)

- Get you laptops ready and login to www.crio.do
- Open [Encapsulation Byte](#) and start your workspace.
- We will be coding away in the session!



Crio Sprint: JAVA-112

OOPs: Abstraction



Today's Session Agenda

- Abstraction
- Abstract Classes
- Interfaces
- Keywords
 - **abstract**
 - **interface**
 - **implements**



Abstraction - Scenario #1 SmartPhone

- We've all made phone calls
- What does a normal user do when (s)he receives a call ?
 - Pick up the call
 - Reject the call
- Does user need to know how the call is created?
 - No, there's a lot of code that runs in the background.
 - Not relevant to the user.
 - That's the beauty of abstraction - you can still operate a phone without knowing how it internally functions.



Abstraction - Scenario #2 ATM

- What operations can be done at an ATM?
 - Cash withdrawal
 - Mini statement
 - Change password, etc.
- List down the steps for cash withdrawal
 - Enter the Card
 - Input the Pin
 - Input Amount for cash withdrawal
 - Validate Amount (whether account has balance)
 - Deduct Amount from Account
 - Dispense Cash



Activity 1 - ATM Machine

```
class ATMMachine {
    public void enterCard(){
        System.out.println ("Card Verification");
    }
    public void enterPin(){
        System.out.println ("Pin Verification");
    }
    public void cashWithdrawal (){
        System.out.println ("To withdraw cash from ATM");
    }
    public void validateWithdrawAmount(){
        System.out.println ("Validate the Amount to be withdrawn");
    }
    public void updateAmount(){
        System.out.println ("Update the Amount after withdrawal");
    }
    public void cashDispense(){
        System.out.println ("Dispense the cash from ATM");
    }
    public void miniStatement () {
        System.out.println ("Get the mini statement");
    }
}
```

Compile and Run the below program.

```
public class Main{
    // Mimic user behavior
    public static void main (String[]args){
        ATMMachine am = new ATMMachine();
        am.enterCard();
        am.enterPin();
        am.cashWithdrawal();
        am.validateWithdrawAmount();
        am.updateAmount();
        am.cashDispense();
    }
}
```

- Look at the output. Does it make sense?
- Remove the below method and run again.
 - validateWithdrawAmount()
 - updateAmount()
- Does it make sense?
- Where should these removed methods be invoked?



Activity 1.1 - ATM Machine

```
class ATMMachine{
    public void enterCard (){
        System.out.println ("Card Verification");
    }
    public void enterPin (){
        System.out.println ("Pin Verification");
    }
    public void cashWithdrawal(){
        System.out.println ("To withdraw cash from ATM");
        validateWithdrawAmount();
        updateAmount();
        cashDispense();
    }
    private void validateWithdrawAmount(){
        System.out.println ("Validate the Amount to be withdrawn");
    }
    private void updateAmount(){
        System.out.println ("Update the Amount after withdrawal");
    }
    private void cashDispense (){
        System.out.println ("Dispense the cash from ATM");
    }
    public void miniStatement (){
        System.out.println ("Get the mini statement");
    }
}
```

Run the program again.

```
public class Main{
    public static void main (String[]args){
        ATMMachine am = new ATMMachine();
        am.enterCard();
        am.enterPin();
        am.cashWithdrawal();
    }
}
```

- Look at the output. Does it make sense?
- What did we accomplish?



What is Abstraction?

- Hide Implementation Complexity
- **Expose only relevant functionality** to the user.
- Focus on ***what the*** object does instead of ***how it*** does it.
- For eg, A computer can connect to a network using (Ethernet, Wi-Fi, dial-up modem, etc.)
 - Web Browser doesn't care which one is being used.
 - **"Connection to the network"** is the **abstraction**.
 - **Ethernet** and **Wi-Fi** are **implementations** which enables connection.

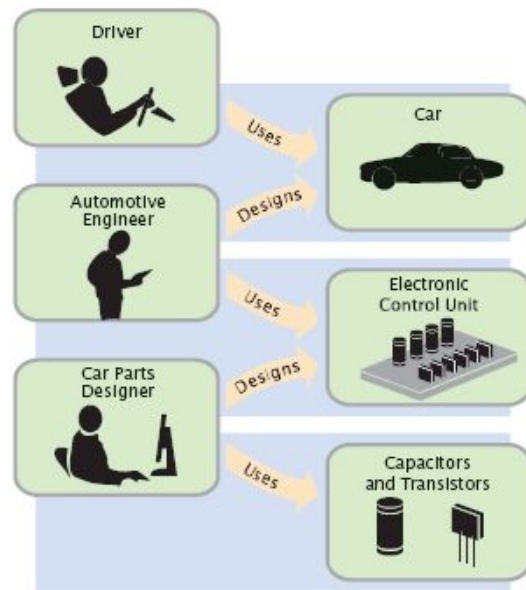


Figure 1

Levels of Abstraction in Automotive Design

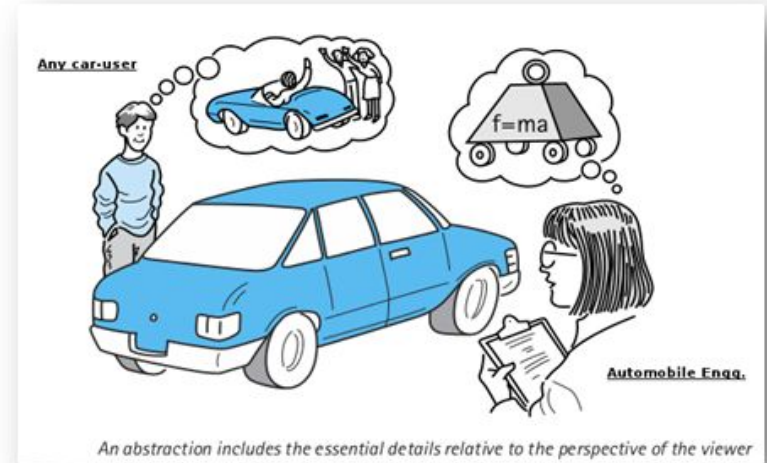
In simple words, **Abstraction** is the practice of **breaking a large problem** into **smaller components**, so **each smaller problem** can be **worked on in (relative) isolation**.



Curious Cats



- How is Abstraction different from Encapsulation?
 - Abstraction refers to
 - **Hide the implementation details or complexity.**
 - **Expose only relevant functionality** to the user.
 - End user focus is on ***what the*** object does instead of ***how it*** does it.
 - By using access specifiers, but there are other ways as well, as we'll see next
 - Encapsulation refers to
 - **Grouping the behavior and data in a single logical unit** and enable **data hiding** to protect the access of data from the outside world.
 - Use of access specifiers



Curious Cats



- Other than making methods *private*, are there other ways to achieve abstraction?
 - **abstract** class (0% - 100% Abstraction)
 - **interface** (100% Abstraction)



Abstract class - Real World Analogy

Real-World Analogy

- **Abstract class** is similar to a **Car Platform** where minimum skeleton is provided.
- The base platform provides bare minimum functionality common between all the cars.
- Different types of car models can be manufactured on top of car platform.
- **Important to Note:-** This skeleton is only useful if it is fully developed into a car.



What is an Abstract class?

- Cannot be instantiated, but they can be subclassed
- May contain regular fields
- May contain **abstract methods**

Syntax of abstract class:

```
public abstract class Animal{
```

We will understand more about it when we solve the next activity

- Declared in **abstract classes**.
- **Body must be empty (no curly braces)**
- It doesn't provide **any actual implementation**

Syntax: `public abstract void build();`



Activity #1 Elementary Exercise - Payroll

- Your manager has asked you to develop a payroll program for a company.
 - The company has two groups of employees
 - **Full-time employees**
 - Gets paid a fixed salary
 - **Hourly Employees**
 - Gets paid by hourly wages
 - The program prints out a payroll that includes employee names and their monthly salaries.
 - To model the payroll program in an object-oriented way, we come up with the following classes:
 - Employee
 - FulltimeEmployee
 - HourlyEmployee
 - and Payroll
- Open this milestone in the Byte to solve this activity
 - [Payroll](#)



Summary - Abstract classes and Methods

- **abstract** keyword is a **non-access modifier**.
- Can be used with **methods** and **classes**.
- Abstract keyword **cannot be used with** following keywords
 - final
 - static
 - private
- Abstract classes **cannot be instantiated**.
- Subclass of an abstract class must
 - **Implement all abstract methods** in the super-class
 - Or **be declared abstract itself**.
- A class that contains at **least one abstract method** must be declared abstract.
- Abstract methods have no body.



Curious Cats



- Why does an abstract class have a constructor even though we cannot create an object?
 - [Why do abstract classes in Java have constructors? - StackOverflow](#)
- Can a non-abstract class have an abstract method?
 - No. A non-abstract class cannot have an abstract method whether it is inherited or declared in Java.
- Should an abstract class have at least one abstract method?
 - No.
- Can abstract methods be made private?
 - No. Private methods(cannot be inherited) and can't be overridden, so they can't be abstract.
- Can we declare an abstract method final or static in java?
 - [Can we declare an abstract method final or static in java? \(tutorialspoint.com\)](#)



5 minute break



Interface - Real World Analogy

Real-World Analogy

- **Interface** is similar to a **Sim Slot** in a mobile phone.
- We can add sim of any Telecom Provider as long as it strictly adheres to the specifications need to fit in the sim slot.
- **Important to Note:-** Sim Slot is an interface which can accept any type of SimCard as long as the SimCard implements the requirements specified by the Sim Slot.



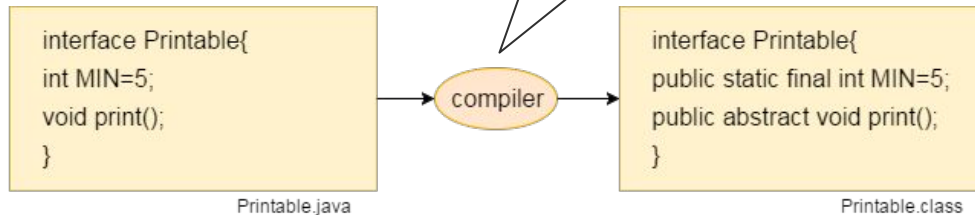
Java Interface

- What is an **interface** and why do we need it?
 - A contract which specifies what a class must do and not how.
 - Anybody can implement this contract and we can easily switch between these implementations.

- How to declare an interface?

```
interface <interface_name>{  
    // declare constant fields  
    // declare methods  
}
```

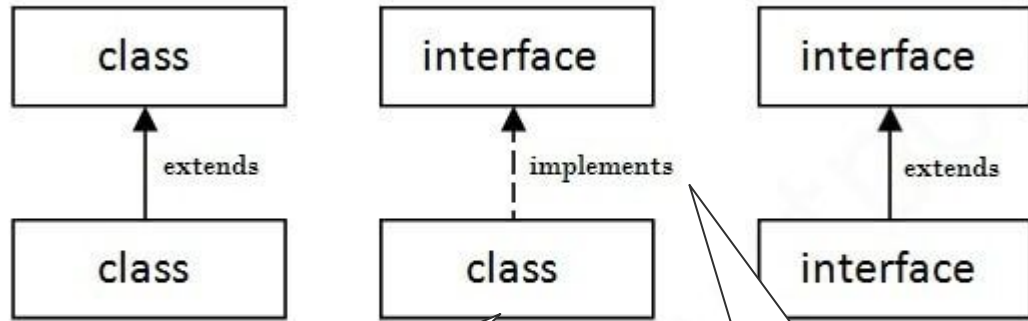
- fields are public, static and final by default
- methods are public and abstract.



We will understand more about it when we solve the next activity



Relationship between Classes and Interfaces



Class which implements interface must provide logic in every method body.

What is implements keyword?










Activity 2 - Amazon checkout Payment

- Have you shopped on Amazon and paid during checkout?
- You will see multiple payment options - Credit card, Net Banking, UPI etc.
- Open this milestone in the Byte to solve this activity
 - [Amazon Payment Options](#)

☐ **Standard Chartered Bank Credit Card** ending in 3545   [Update](#)

☐ **Citibank Debit Card** ending in 0901    [Update](#)

Another payment method

- ☐ **Add Debit/Credit/ATM Card**
- You can save your cards as per new RBI guidelines. [Learn More](#) ▾
-       
- ☐ **Net Banking**
-
- ☐ **Other UPI Apps**



Curious Cats



- Can we achieve multiple inheritance in Java?
 - Not possible with classes.
 - But,

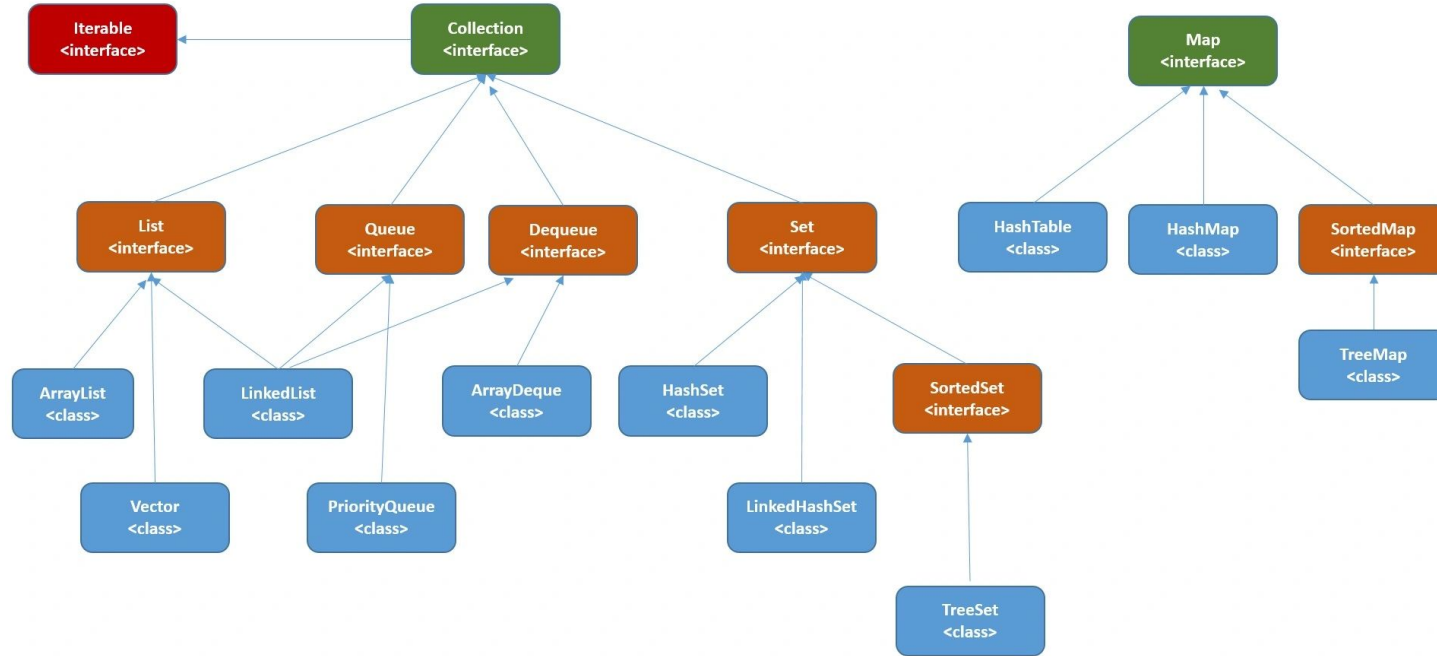


Multiple Inheritance in Java



Application of Interfaces in Java Collection

Collection Framework Hierarchy



Summary - Interfaces

- An **interface** defines a contract which specifies what a class **must do** and **not how**.
- An interface declaration contains
 - **Signatures, but no implementations** for methods
 - May also contain constants
- A class that **implements an interface** must implement all the **methods declared in the interface**.
- You can use interface names anywhere you can use any other data type name. If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface.
- **Multiple inheritance** can be achieved using *interfaces*.



Curious Cats



- How is an interface different from an abstract class? When to use which?
 - Detailed Reading:- [Difference between Abstract Class and Interface in Java - GeeksforGeeks](#)



Abstraction Exercises Byte Overview



Overview: Reinforcement Exercise - Google Form

[Reinforcement Exercise - Google Form](#)



Overview: Challenge Exercise - Shape of Shapes

[Challenge Exercise - Shape of Shapes](#)



Overview: Challenge Exercise - Number Game

[Challenge Exercise - Number Game](#)



Questions

1. What is abstraction in object-oriented programming?
2. What is an abstract class in Java?
3. What is the difference between an abstract class and an interface in Java?
4. What is the purpose of interfaces in Java?
5. What is the purpose of the "implements" keyword in Java, and how is it used?



Session Revision Quiz

[Quiz Link](#)

Solve this quiz to access your understanding of session's topics clearly



Take home exercises for the session

- [Abstraction Byte](#)
 - [Abstraction Quiz](#) (Link Present in Byte)

All of these details are also available on the site.



Further Reading

- [Java Abstract Keyword - Javatpoint](#)
- [Oracle docs - Abstract](#)
- [Oracle docs - Interface](#)
- [Abstract Classes and Abstract Methods in Java - Dot Net Tutorials](#)
- [Abstraction in Java | Abstract Class, Example - Sciencetech Easy](#)
- [OOP Case Study I/O](#)
- [Lab Exercise: Abstraction and Encapsulation](#)



Thank you

