תכנות מונחה עצמים
תרגול 01

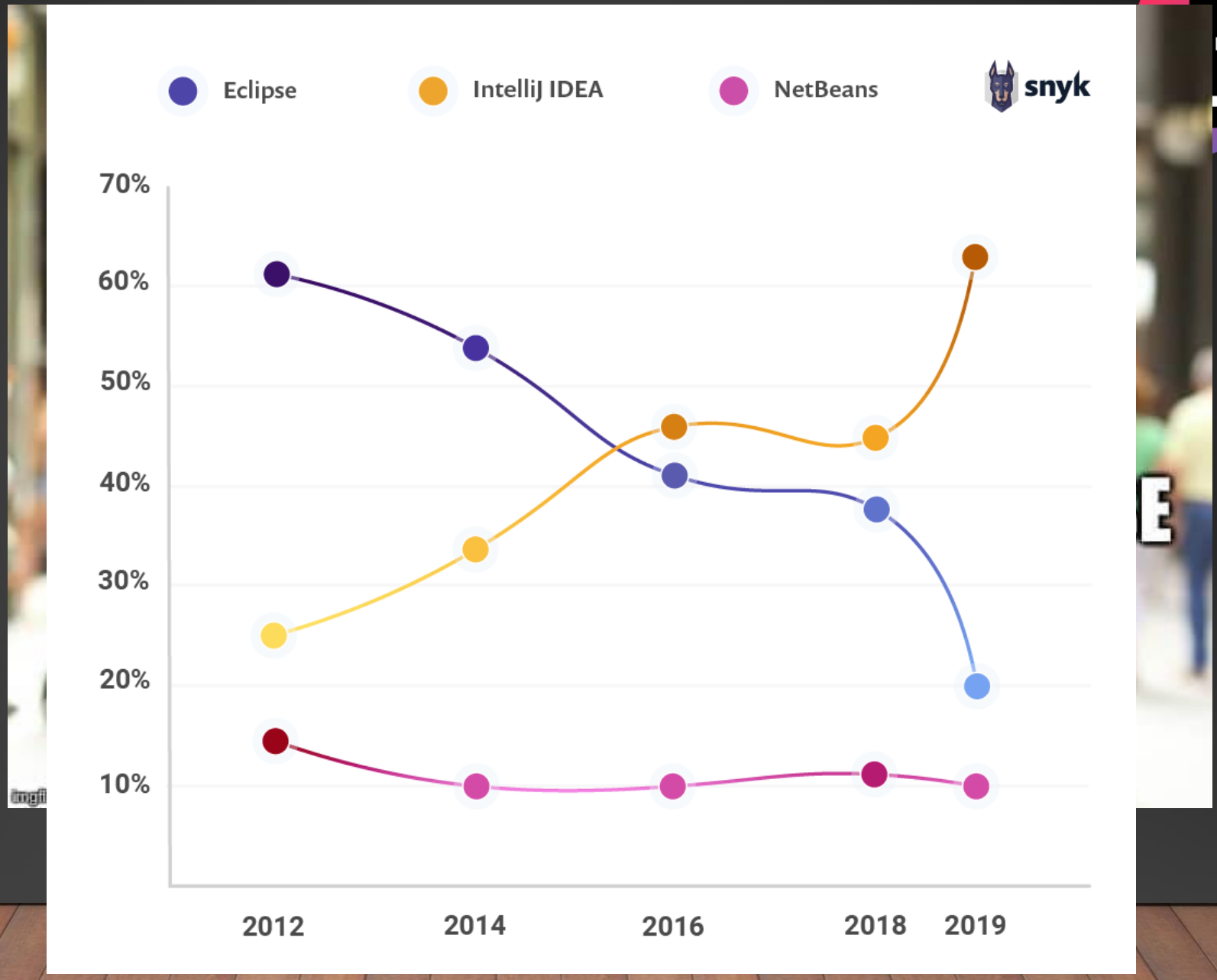
מייל: Netanel.levine@msmail.ariel.ac.il
נכתב ע"י: נתנאל לוין

נושאים להיום

- בקצרה על הקורס .
- קצת על JAVA
- יישור קו.
- מושג האובייקט המחלקה .
- שיטות ת הרשאות .
- וירושה.
- ממשקים
- איך מתחילים את המטלה.

כמה מילים על הקורס:

- מה אתם מקבלים מהקורס הזה?
  - **לתכנת זה לא רק לכתוב קוד -** אחד ההבדלים בין מתכנת לבין מישהו שיודע לכתוב קוד זה העקרונות שאתם תלמדו כאן.
  - בין הנושאים הנפוצים ביותר שישאלו אתכם בראיונות עבודה.
  - להתחיל לפתח את תיק העבודות שלכם בגיטהאב.
- למה מצפים ממכם?
  - **שתלמדו לבד ומעבר לסקופ של הקורס** – פה אתם תלמדו את העקרונות הבסיסיים של OOP, יש עוד כל כך הרבה ללמוד.
  - יושר (תהיה בדיקת העתקות) ולהיצמד להוראות של המטלות.
  - להשקיע במטלות הם שוות 50% מהציון של הקורס.
  - להבין את התאוריה של מה שאתם עושים.
  - ללמוד לחפש מידע.
- איפה ניתן לראות את החומרים של ההרצאה והתרגול ואיך עושים את זה ?
- כל הקודים נמצאים ב GitHub
- קישור


Eclipse
IntelliJ IDEA
NetBeans
snyk
70%
60%
50%
40%
30%
20%
10%
2012
2014
2016
2018
2019

GOOGLE IS YOUR BEST FRIEND FROM NOW !

How to syso in IntelliJ ????

Answer

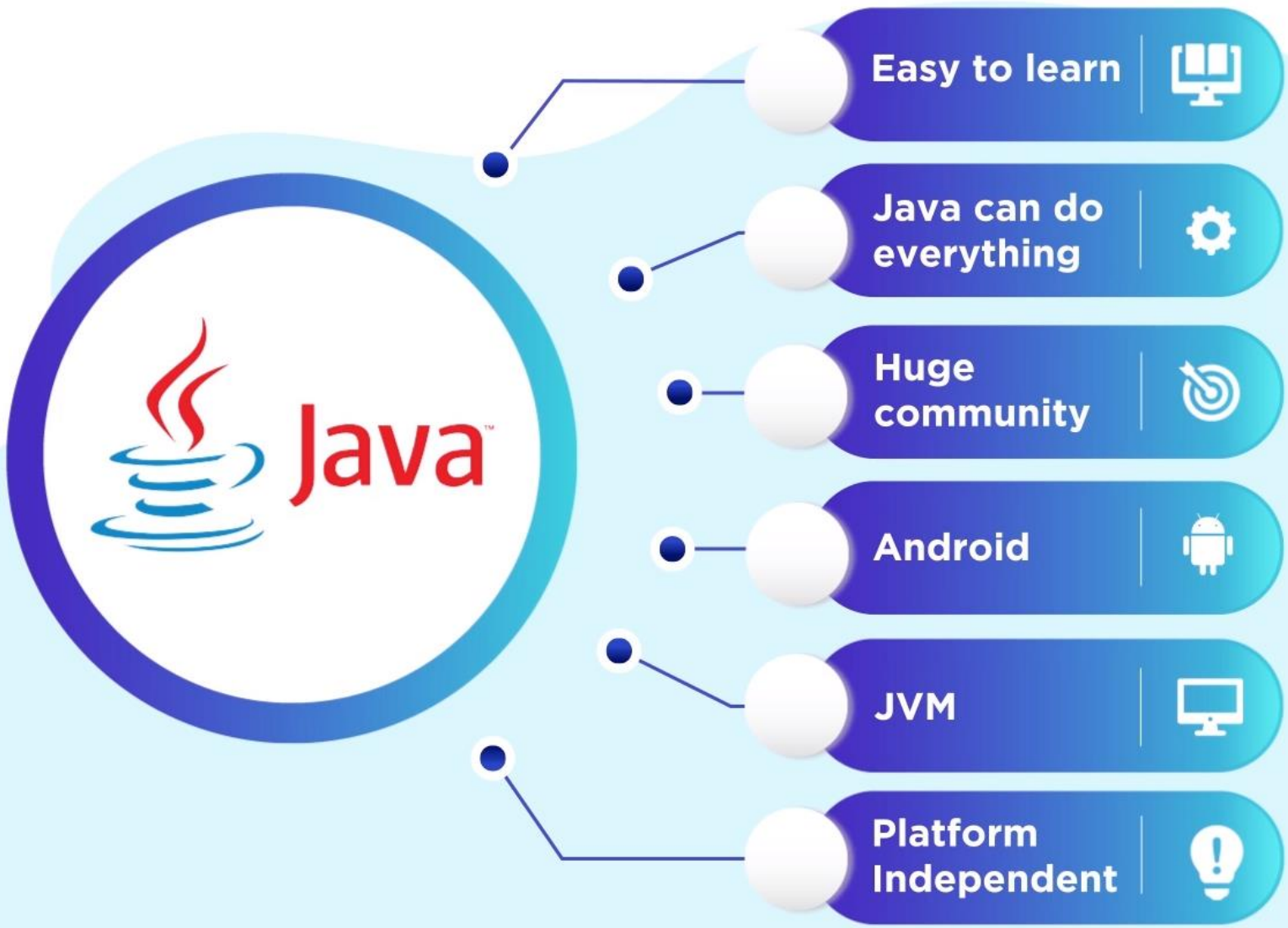
מה זה JAVA

ג'אווה היא שפת תכנות מונחית עצמים.
לרוב עוברות תוכניות ג'אווה הידור (**קימפול**) ל־Java bytecode, שפת ביניים דמוית שפת מכונה, שאותה מריצה מכונה וירטואלית (Java Virtual Machine; **JVM**).

ג'אווה היא שפה בעלת טיפוסיות **סטטית חזקה**, כלומר לכל ביטוי בשפה מתאים טיפוס יחיד, תקינות ביטויים נבדקת בזמן ההידור (קימפול), במידת האפשר. כאשר אין זה אפשרי, תתבצע בדיקה בזמן ריצה.
לכל טיפוס פרימיטיבי בג'אווה יש מחלקה עוטפת, אליה וממנה מתבצעת המרה אוטומטית.
למשל, לטיפוס הפרימיטיבי int קיימת מחלקה עוטפת Integer.

שפת ג'אווה כוללת גם **ניהול זיכרון אוטומטי**.
המתכנת פטור מן ההכרח לשחרר זיכרון המוקצה לאובייקט ברגע שאין עוד משתנים המצביעים עליו.
במקום זאת, סביבת זמן הריצה כוללת מנגנון "איסוף זבל" (**garbage collector**), המבצע זאת אוטומטית.
ג'אווה מאפשרת **ירושה יחידה בלבד**, וזאת על מנת למנוע בעיות דו־משמעות הנוצרות מירושת יהלום.
כדי לאפשר גמישות דומה לזו של ירושה מרובה, קיים בג'אווה מנגנון **ממשק** (**interface**) המגדיר רשימה של מתודות, המהווה חוזה עם המתכנת.
כך כל מחלקה יכולה **להרחיב** (**extends**) לכל היותר מחלקה אחת, **ולממש** (**implements**) מספר בלתי מוגבל של ממשקים.
על המחלקה לממש במפורש כל מתודה בכל ממשק כזה, וכך לא קיימת בעיה של כפל משמעות.
ג'אווה תומכת **בתכנות גנרי** (**Generics**), מספר משתנה של פרמטרים לפונקציה ו**טיפוסי מניה** (**Enums**).

Why Java?


Java
Easy to learn
Java can do everything
Huge community
Android
JVM
Platform Independent

JVM, JRE & JDK

Java Virtual Machine

This is program written in C++
which is responsible for converting
Byte code to machine specific code.

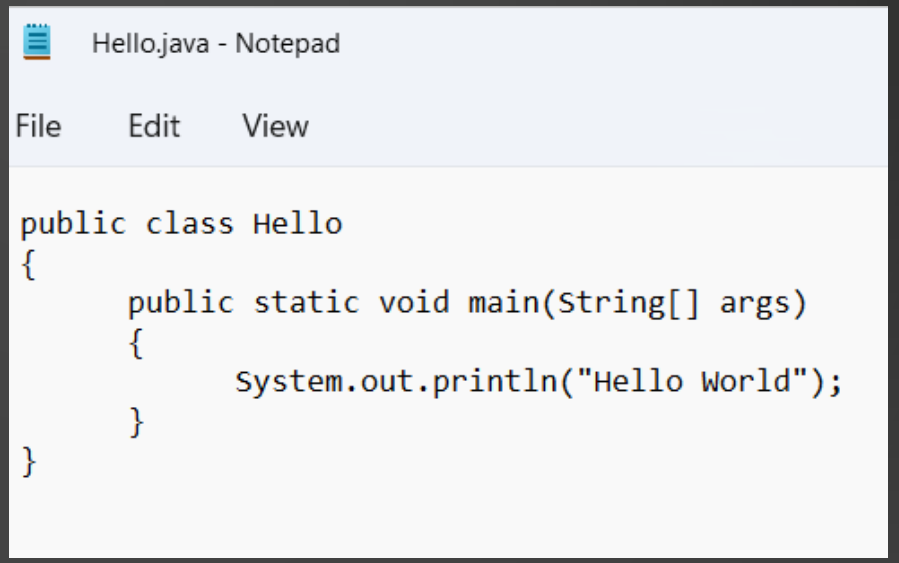
You compile your source code to java bytecode --> you execute your bytecode on JVM.

The JVM is used for both — translate the bytecode into the machine language for a particular computer, and actually execute the corresponding machine-language instructions as well.

```
C:\Users\netan>javac Hello.java

C:\Users\netan>java Hello
Hello World

C:\Users\netan>
```

The Java programming language compiler, javac, reads source files written in the Java programming language, and compiles them into bytecode class files, after this line was executed javac created Hello.class file.

The java command starts a Java application. It does this by starting the Java Virtual Machine (JVM), loading the specified class, and calling that class's main() method.



JVM, JRE & JDK

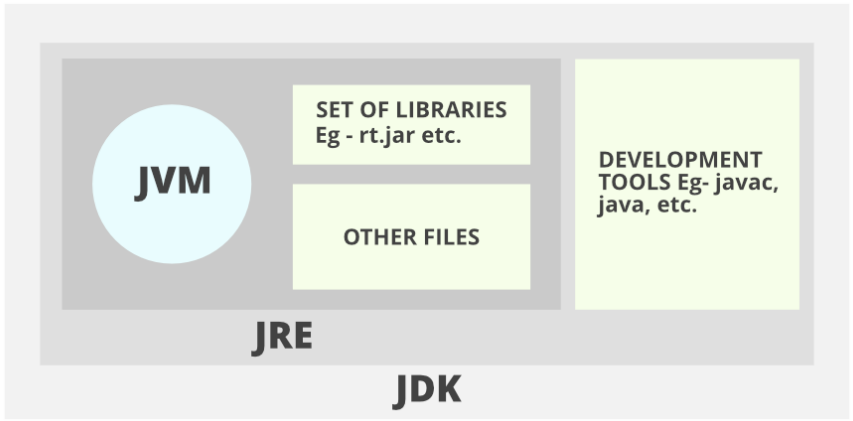
Java Runtime Environment

JRE includes JVM in itself PLUS java binaries and classes which are needed to execute program.

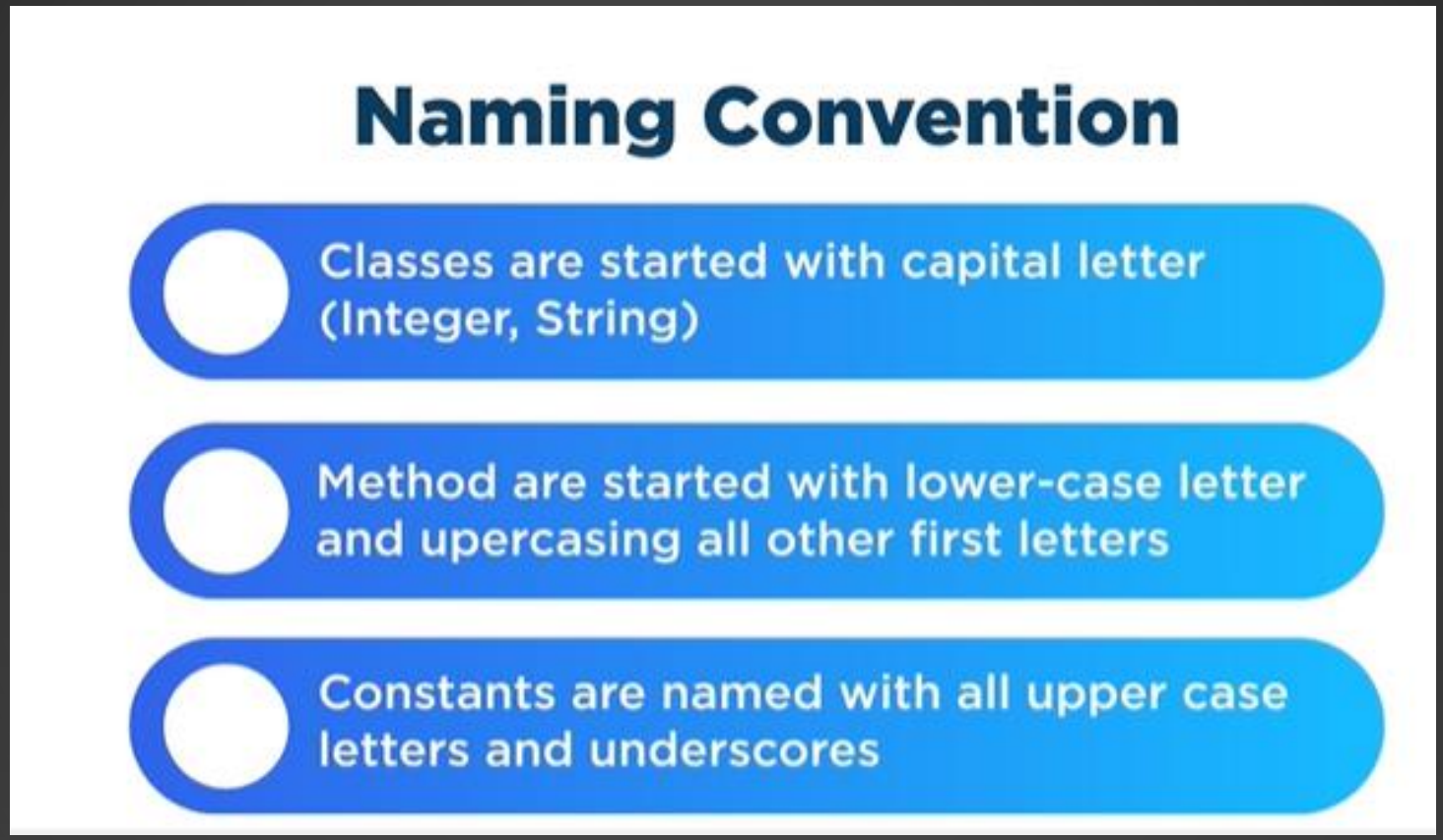
JVM, JRE & JDK

Java Development Kit

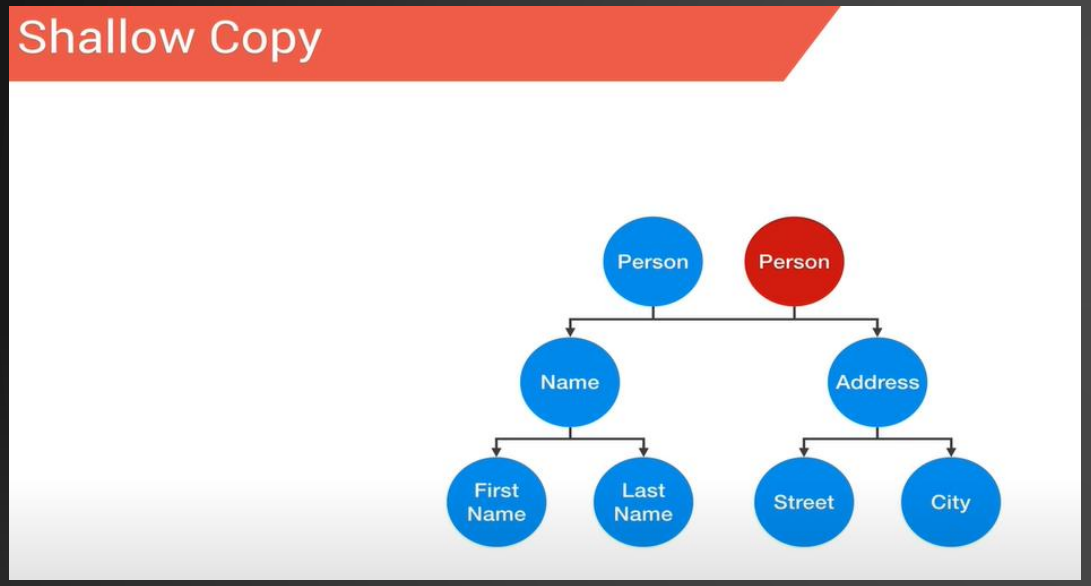
JDK contains JVM and JRE PLUS all tools which are necessary for development of Java programs

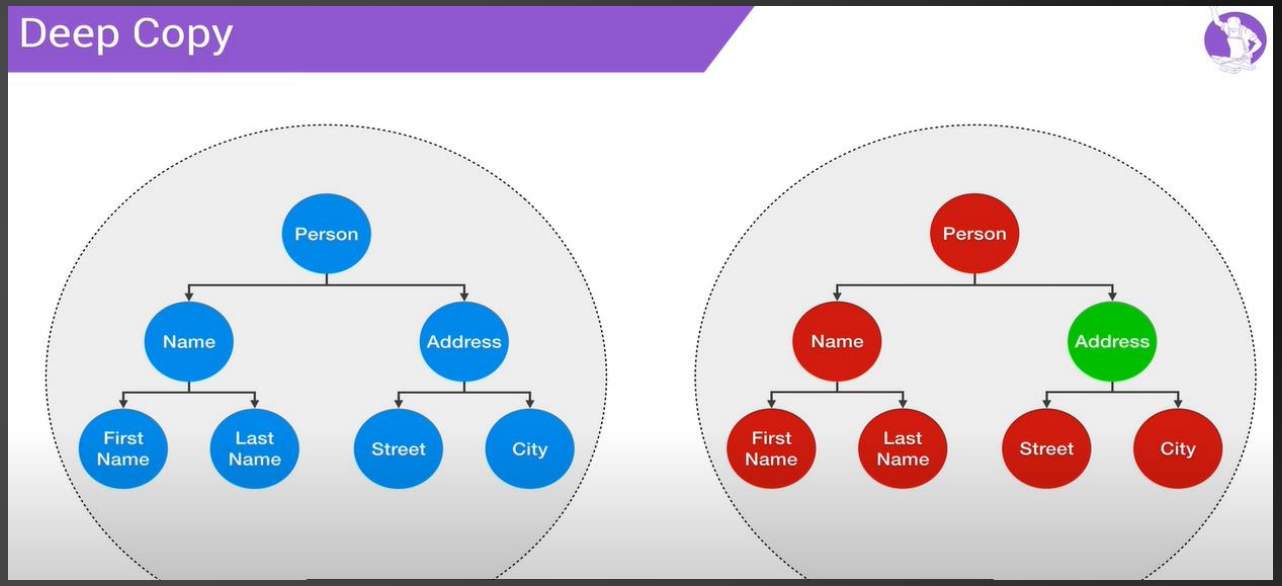


NAMING CONVENTIONS



יישור קו
העתקה עמוקה (DEEP COPY)





https://www.youtube.com/watch?v=QaCYMgyprtc

המושג גנרי

תכנות גנרי - הוא סגנון פיתוח תוכנה בו אלגוריתמים נכתבים במונחים של טיפוסים שיוגדרו בהמשך, וסוג הטיפוס יוגדר על פי הצורך ואז יתבצע שימוש באלגוריתם עבור הטיפוס המסוים שנקבע.
(ויקיפדיה)

דוגמא למחלקה גנרית שכולכם מכירים :

```
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

```
1  public class Main {
2      public static <T> void swap(T[] array, int left, int right){
3          T temp = array[right];
4          array[right] = array[left];
5          array[left] = temp;
6      }
7  }
```

SHORTEN IF

The value of a variable often depends on whether a particular Boolean expression is or is not true and on nothing else.

For instance, one common operation is setting the value of a variable to the maximum of two quantities. In Java you might write

```
if (a > b) {
    max = a;
}
else {
    max = b;
}
```

Setting a single variable to one of two states based on a single condition is such a common use of if-else that a shortcut has been devised for it

, the conditional operator, ?:. Using the conditional operator you can rewrite the above example in a single line like this:

```
max = (a > b) ? a : b;
```

FOR EACH

Out:

```
String[] stringArr = {"\nThis","is","for","each",":)\n"};
for (String s : stringArr) {
    System.out.print(s+" ");
}
```

```
This is for each :)
```

ENUM

An Enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).

To create an Enum, use the Enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

Example

```
public enum Priority {
        HIGH, MEDIUM, LOW;
}
```

Example

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

You can access `enum` constants with the **dot** syntax:

```
Level myVar = Level.MEDIUM;
```

ENUM

```
public class EnumDemo {

	public static void main(String[] args) {

		Priority priority = Priority.HIGH;

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

		System.out.println("============ Enum valueOf()");

		Priority priority2 = Priority.valueOf("HIGH");
		System.out.println(priority2);

//		priority2 = Priority.valueOf("high"); // java.lang.IllegalArgumentException: No enum constant com.itbulls.learnit.javacore.enumerations.Priority.high
```

ENUM

```
System.out.println("============ Enum comparison");

System.out.println("Priority.HIGH == Priority.MEDIUM: "
                                            + (priority == Priority.MEDIUM));       // false

System.out.println("Priority.HIGH == Priority.HIGH: "
                                            + (priority == Priority.HIGH)); // true



System.out.println("============ Enum ordinal()");

System.out.println("Priority.HIGH.ordinal(): " + Priority.HIGH.ordinal());
System.out.println("Priority.MEDIUM.ordinal(): " + Priority.MEDIUM.ordinal());


System.out.println("============ Enum iteration");

Priority[] values = Priority.values();
for (Priority priority3 : values) {
        System.out.println(priority3);
}
```

ENUM

```
public enum Month {

        JANUARY(31), FEBRUARY(28), MARCH(31), APRIL(30), MAY(31), JUNE(30), JULY(31),
        AUGUST(31), SEPTEMBER(30), OCTOBER(31), NOVEMBER(30), DECEMBER(31);

        private int daysAmount;

        private Month(int daysAmount) {
                this.daysAmount = daysAmount;
        }

        public int getDaysAmount() {
                return this.daysAmount;
        }

}
```

```
   System.out.println("============ Enum fields and methods");

   System.out.println("Month.JANUARY.getDaysAmount(): " + Month.JANUARY.getDaysAmount());
```

Agenda

- What is OOP?
- What is an object?
- Class
- Why do we need OOP?
- OOP advantages
- OOP vs Functional programming
- Inheritance
- Encapsulation
- Polymorphism
- Abstraction

Object-oriented programming

Definition

OOP is a programming paradigm based on the concept of 'object'

OOP is a computer programming model that organizes software design around the objects, rather than functions

Object-oriented programming is a software design approach in which concept of 'object' occupies the first place.

What is an object?

Properties

- Doors
- Body
- Trunk
- Steering wheel
- Headlight
- Wheels
- Engine
- Color

What is an object?

Properties

- Doors
- Body
- Trunk
- Steering wheel
- Headlight
- Wheels
- Engine
- Color

What is an object?

Properties

- Doors
- Body
- Trunk
- Steering wheel
- Headlight
- Wheels
- Engine
- Color

Behavior

- Turn on headlights
- Stop engine
- Turn left
- Brake
- Start engine
- Turn right
- Accelerate
- Open trunk

What is an object?

Properties

- Doors
- Body
- Trunk
- Steering wheel
- Headlight
- Wheels
- Engine
- Color

Behavior

- Turn on headlights
- Stop engine
- Turn left
- Brake
- Start engine
- Turn right
- Accelerate
- Open trunk

What is an object?

אובייקט הוא משתנה מטיפוס המחלקה, אנו בונים אובייקט ע"י אתחול משתני העצם של המחלקה.
אובייקט הוא יחידת תוכנה שמספקת שירותים (methods) מסוימים.

Object, first of all, is a virtual entity with specific list of properties which can distinguish it from other objects and behavior which allow to manipulate with these properties

OBJECT

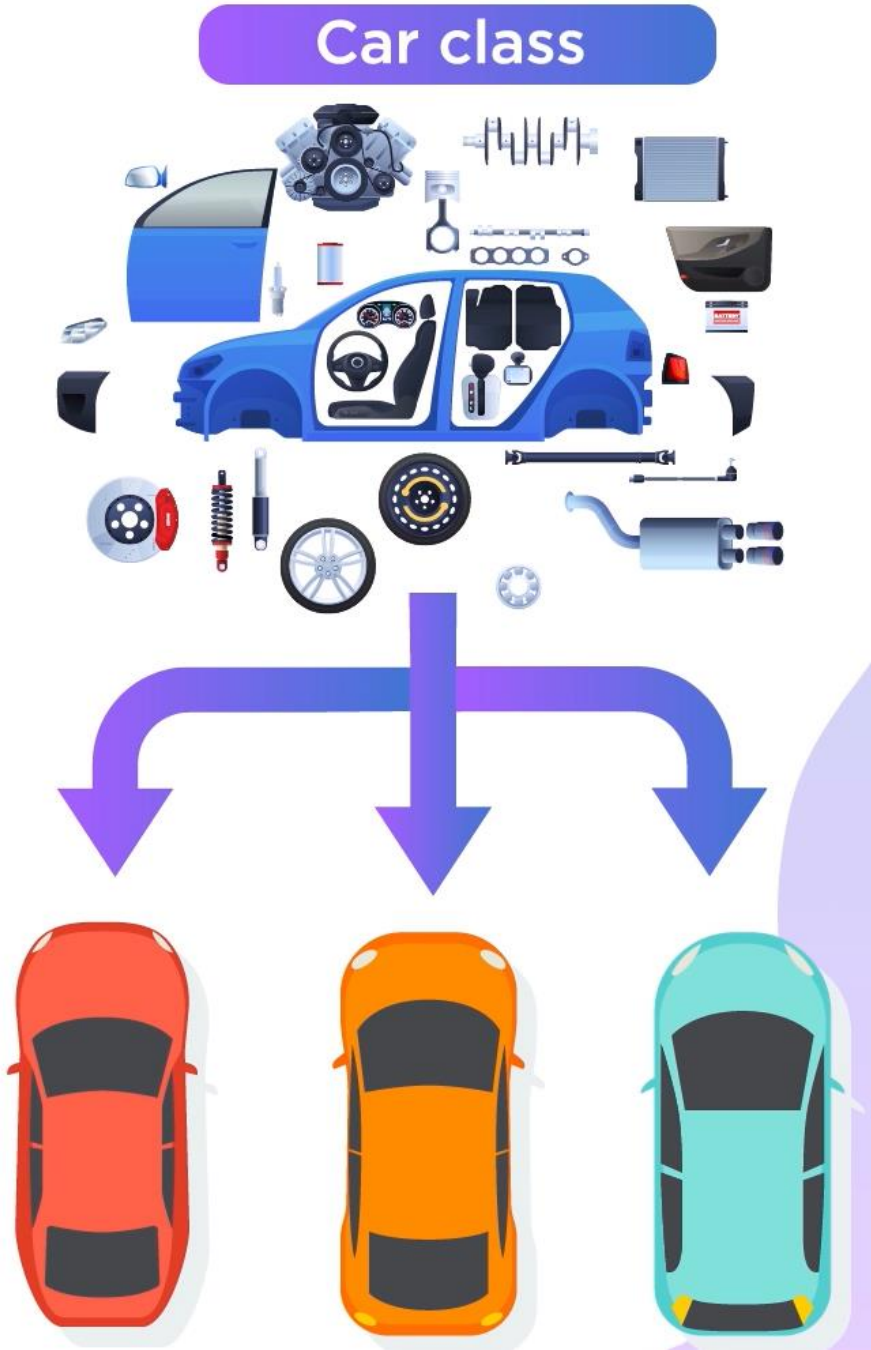
DATA + **BEHAVIOR**

What is a class?

קבוצה של עצמים מאותו סוג, כלומר שמספקים את אותם שירותים באותה הצורה.
מבנה לוגי המאגד בתוכו פונקציות ומשתני עצם תחת שם אחד, ממחלקה ניתן ליצור אובייקטים,
שם המחלקה הוא שם של טיפוס חדש, המוגדר ע"י משתמש.

אובייקטים מול מחלקות
האובייקטים הם מופעים (instances) של המחלקה.
המחלקה היא הישות **הסטטית** בקוד המקור והאובייקט הוא הישות **הדינמית** בזמן הריצה.

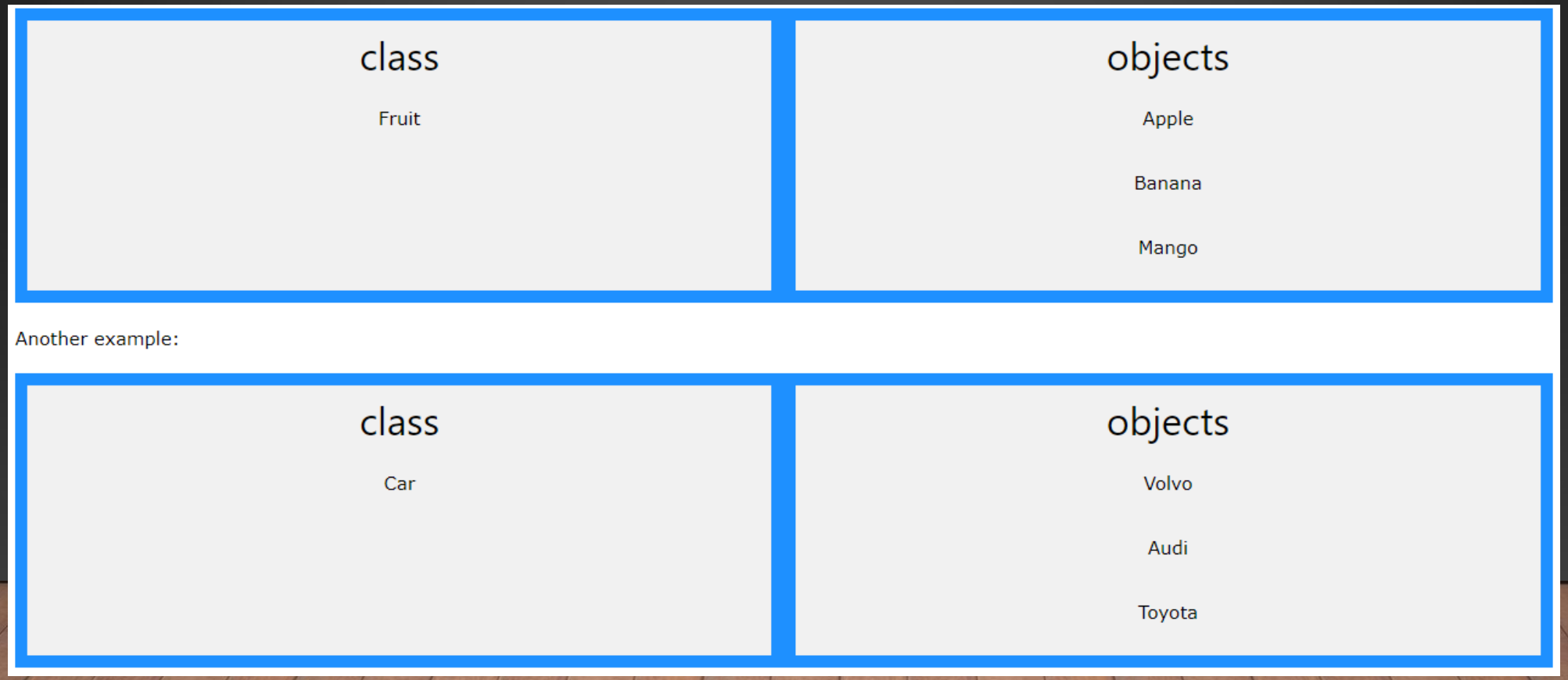
Class is a template for objects



CLASSES

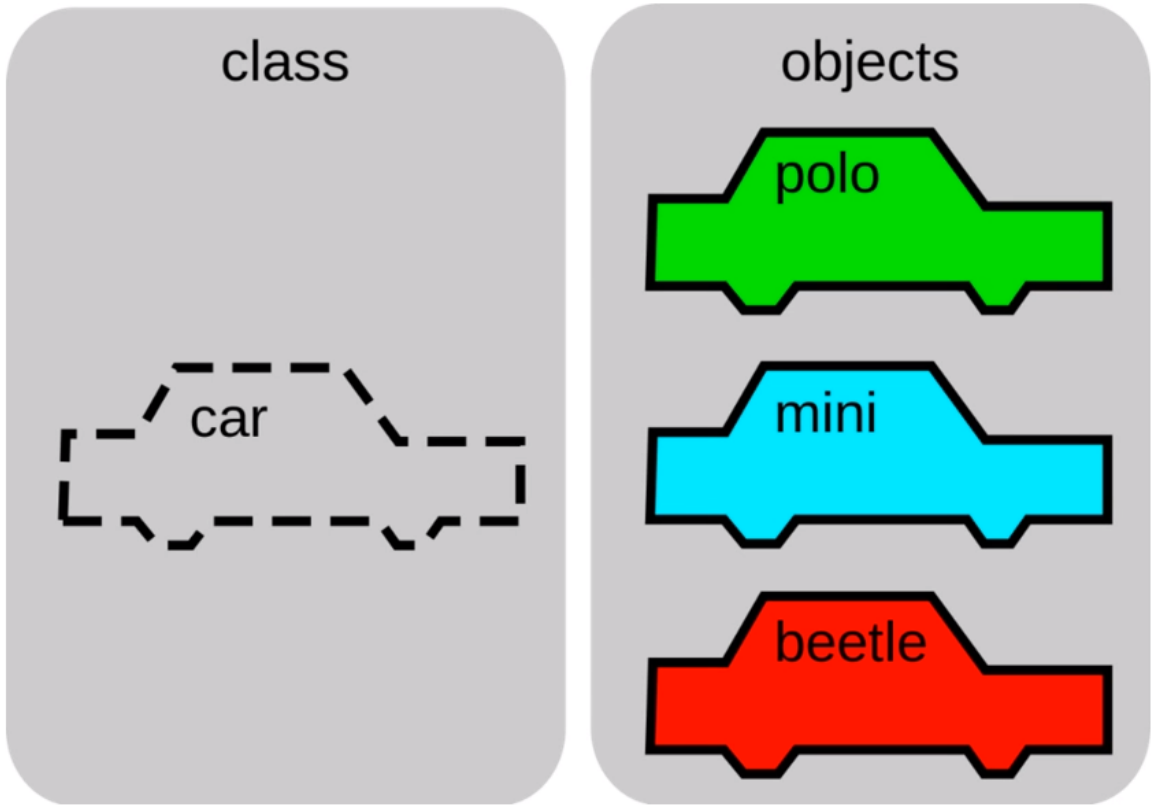
Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



What does class consist off?

- Constructors
- Fields
- Methods
- Initialization blocks
- Nested classes



CLASSES CONSTRUCTORS

A constructor in Java is a **special method** that is used to initialize o*b*jects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

```
public class Product {

        private String name;
        private BigDecimal price;

        public Product() {
        }

        public Product(String name, BigDecimal price) {
                this.name = name;
                this.price = price;
        }
```

CLASSES METHODS

Java class methods:

```
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public BigDecimal getPrice() {
        return price;
    }
    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Product [name=" + name + ", price=" + price + "]";
    }
}
```

INITIALIZATION BLOCK

In a Java program, operations can be performed on methods, constructors, and initialization blocks. Instance Initialization Blocks or IIBs are used to initialize instance variables. So firstly, the constructor is invoked, and the java compiler copies the instance initializer block in the constructor after the first statement super(). They run each time when the object of the class is created.

- Initialization blocks are executed whenever the class is initialized and before constructors are invoked.
- They are typically placed above the constructors within braces.
- It is not at all necessary to include them in your classes.

```
public class Cart {

        private static final int DEFAULT_CART_CAPACITY = 10;
        private static final int MONEY_SCALE = 2;
        private static final double DEFAULT_TAX_RATE = 0.15;
        private static final String DEFAULT_TAX_TYPE = "incomeTax";
        private static final double DEAFULT_DISCOUNT_RATE = 0;
        private static final String DEFAULT_DISCOUNT_NAME = "zeroDiscount";

        private static int cartCounter;

        private int id;
        private int userId;
        private BigDecimal totalNetPrice;               // without taxes
        private BigDecimal totalGrossPrice;     // with taxes
        private BigDecimal totalTax;
        private Tax tax;
        private Product[] products;
        private int indexToAddNewProduct;
        private Discount discount;

        static {
                System.out.println("Cart.class is uploaded into JVM");
        }

        {
                cartCounter++;
                userId = 1;
                tax = new Tax(DEFAULT_TAX_TYPE, DEFAULT_TAX_RATE);
                discount = new Discount(DEFAULT_DISCOUNT_NAME, DEAFULT_DISCOUNT_RATE);
        }
```

Functional programming vs OOP

Functional programming vs OOP

OOP Advantages

- Modularity
- Scalability
- Lower cost of development
- Security & Reliability

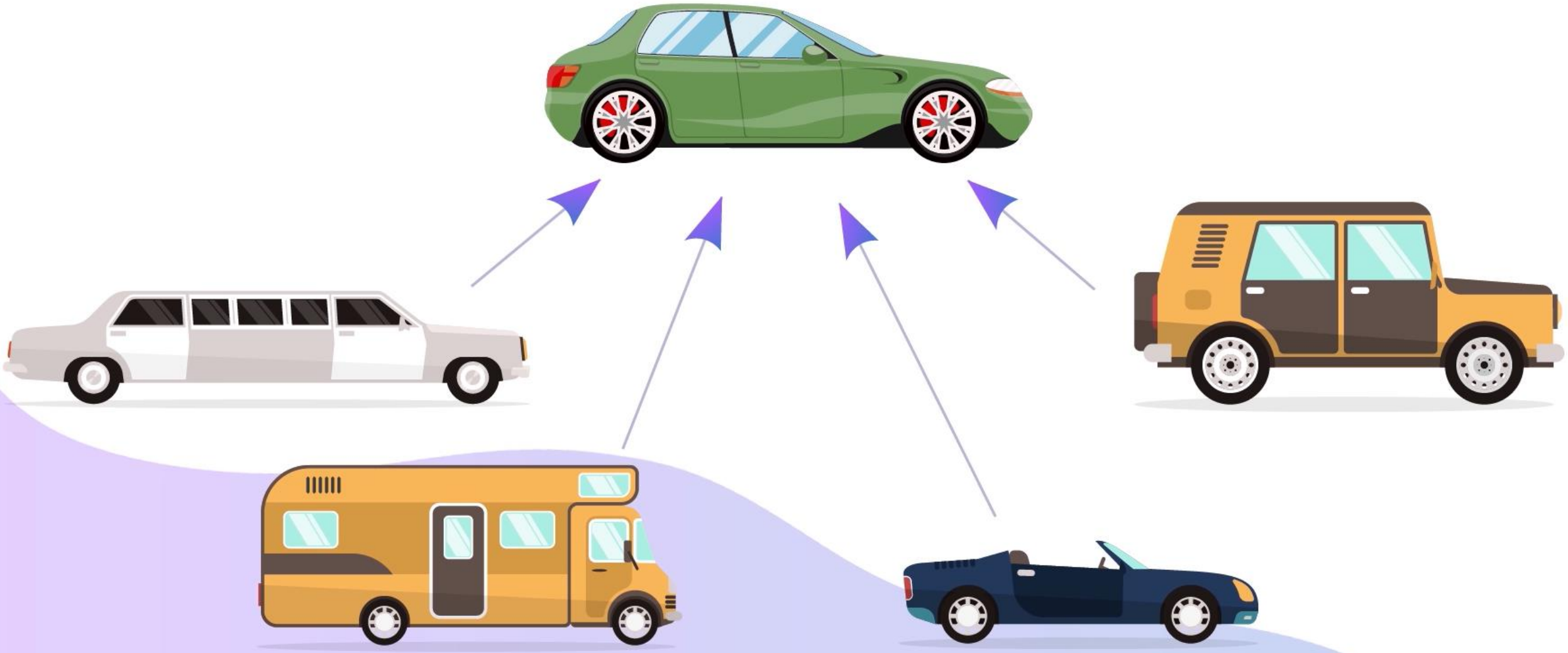
OOP Advantages

Programming is always just a question of trade-offs between maintainability, scalability, speed of development, code quality and reliability.

Basic OOP principles

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

Inheritance

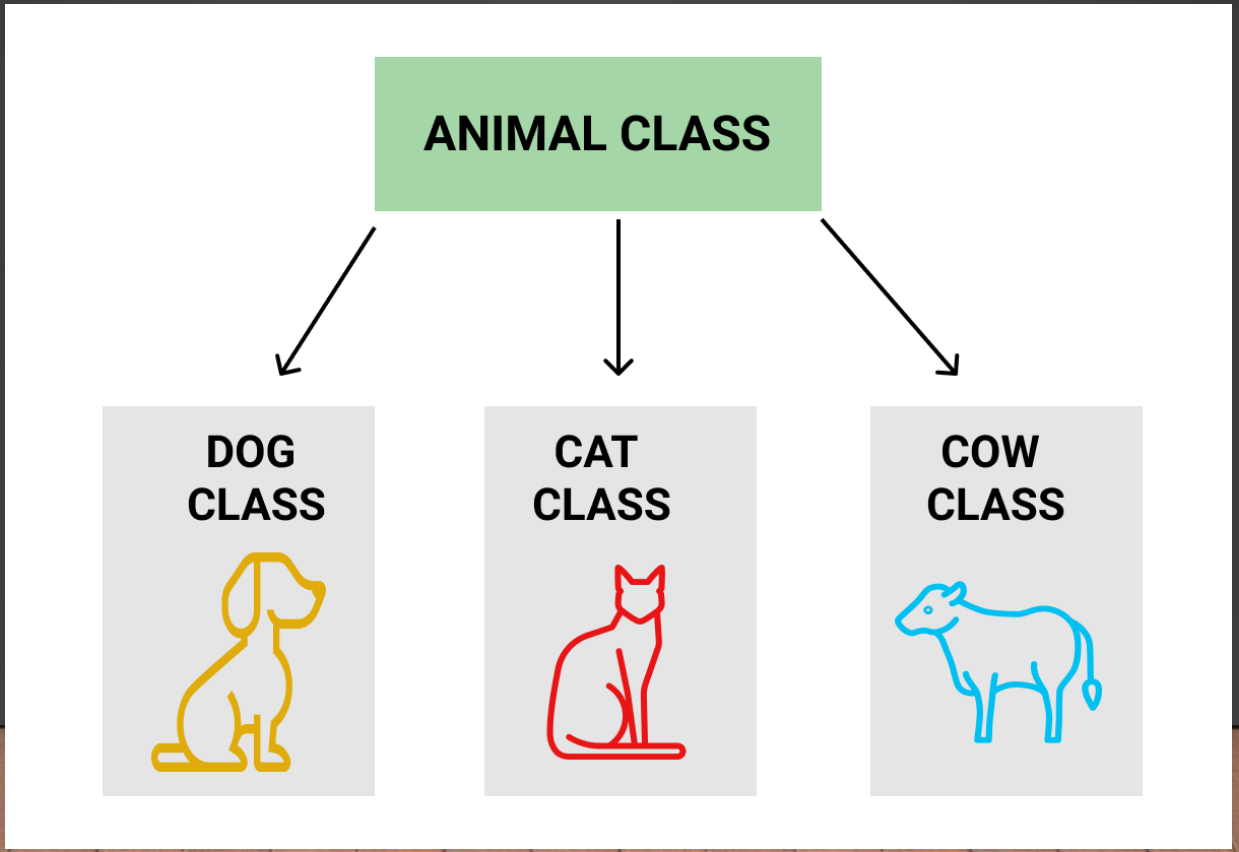


Inheritance

- Inheritance allows you to create new class based on another one.
- New class can extend existing one and share properties and behavior.
- New class is called child class and basic class is called parent class.
- Inheritance allows you to reuse code and to create new classes without reinventing the wheel.

JAVA INHERITANCE (SUBCLASS AND SUPERCLASS)

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from



INHERITANCE

In Java, it is possible to inherit attributes and methods from one class to another.

We group the "inheritance concept" into two categories:

To inherit from a class, use the extends keyword.

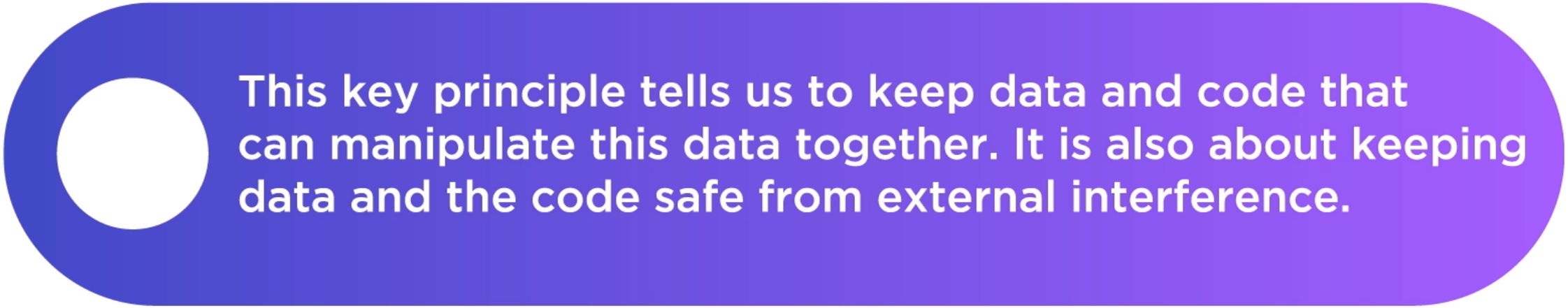
class Subclass-name extends Superclass-name

```java
public class Product {

        private int id;
        private String name;

        public Product(String name) {
                this.name = name;
        }

        public int calculateRemainingAmount() {
                return 100; // just a stub for the sake of example
        }

        public String getName() {
                return this.name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public Product[] listVariants() {
                // some code that fetches the variant products from database
                return new Product[3]; // just a fake return object for the sake of example
        }
}
```

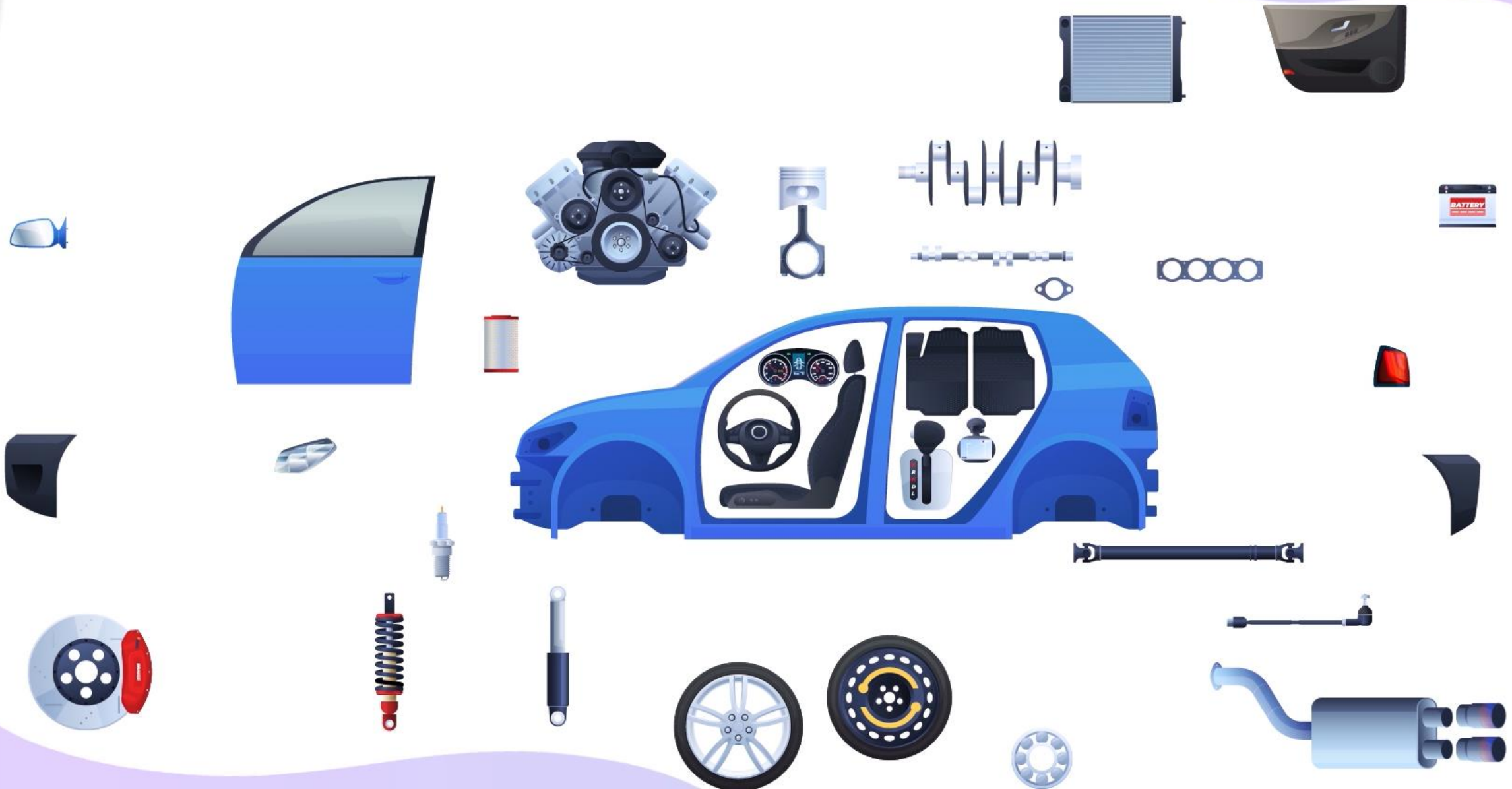
INHERITANCE

```java
public class Phone extends Product {

        // Default constructor
        public Phone() {
                super("Nokia");
                System.out.println("Some code");
        }

        public void ring() {
                System.out.println("Ring!");
        }

        @Override
        public Product[] listVariants() {
                throw new UnsupportedOperationException();
        }

        public int calculateAmountOfVariants() {
                return super.listVariants().length;
        }

}
```

Encapsulation



Encapsulation



CLASSES ENCAPSULATION

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

declare class variables/attributes as private

provide public get and set methods to access and update the value of a private variable

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

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

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

CLASSES PACKAGES & API

Java Packages & API

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

Built-in Packages (packages from the Java API)

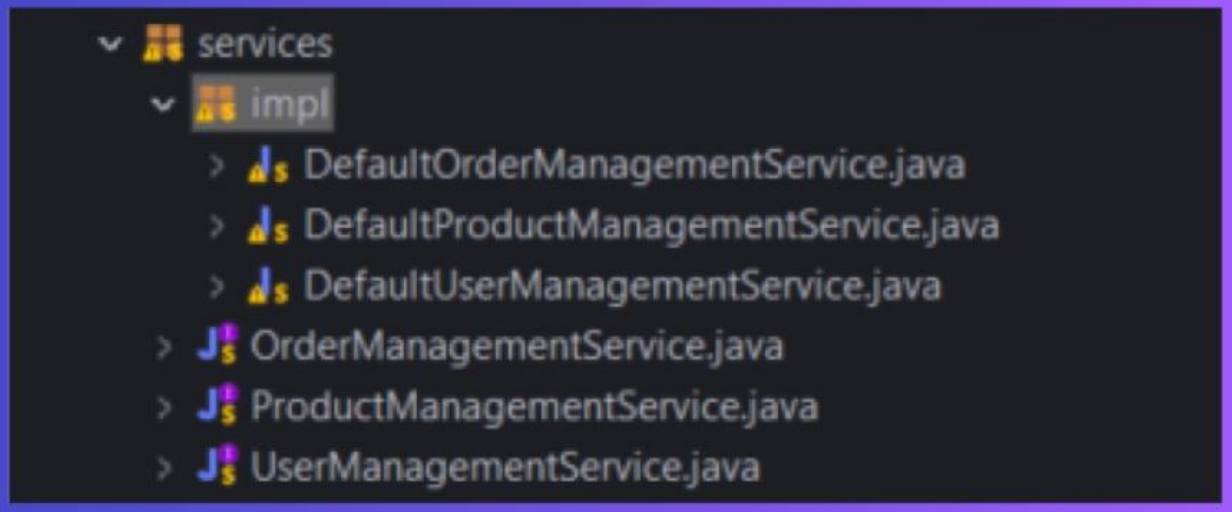
User-defined Packages (create your own packages)

Syntax

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

Encapsulation

- Private
- Default
- Protected
- Public



CLASSES MODIFIER

Modifier	Description
`public`	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter

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

Modifier	Description
`public`	The code is accessible for all classes
`private`	The code is only accessible within the declared class
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
`protected`	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter

Non-Access Modifiers

For **classes**, you can use either `final` or `abstract`:

Modifier	Description
`final`	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)
`abstract`	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)

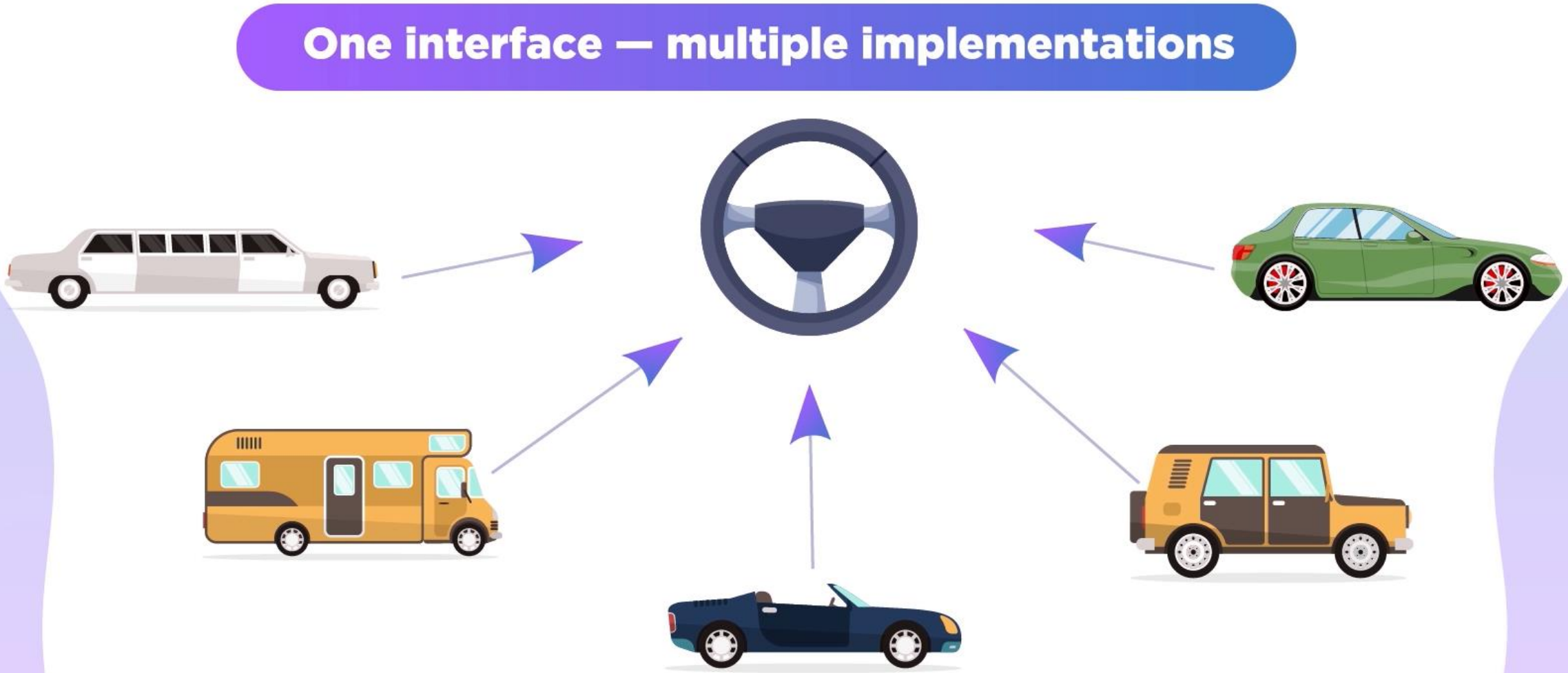
Polymorphism

VS

GASOLINE

ELECTRIC

Polymorphism



POLYMORPHISM

Java Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Abstraction

Abstraction

If you create good abstraction for your specific case it would allow you to not duplicate code, to support good scalability of your units and lots more.

ABSTRACTION

Abstract Classes and Methods

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

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract :

keyword is a non-access modifier, used for classes and methods

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

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

```
Animal myObj = new Animal(); // will generate an error
```

ABSTRACTION

```
public abstract class Product {

        private int id;
        private String name;
        private int minOrderQuantity;
        private boolean isDeliveryAvailable;

        public abstract boolean isAvailableInStock();

        public int getRemainingAmountInStock() {
                // here goes some code which checks amount of product in DB
                return 100; // just a stub for the sake of example;
        }

        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
```

Abstract Class VS Interface

Abstract Class	Interface
‘extends’ keyword	‘implements’ keyword
Can have fields with all possible modifiers	All fields are constants
Can’t be extended simultaneously with other classes	Can be implemented together with other interfaces

INTERFACES

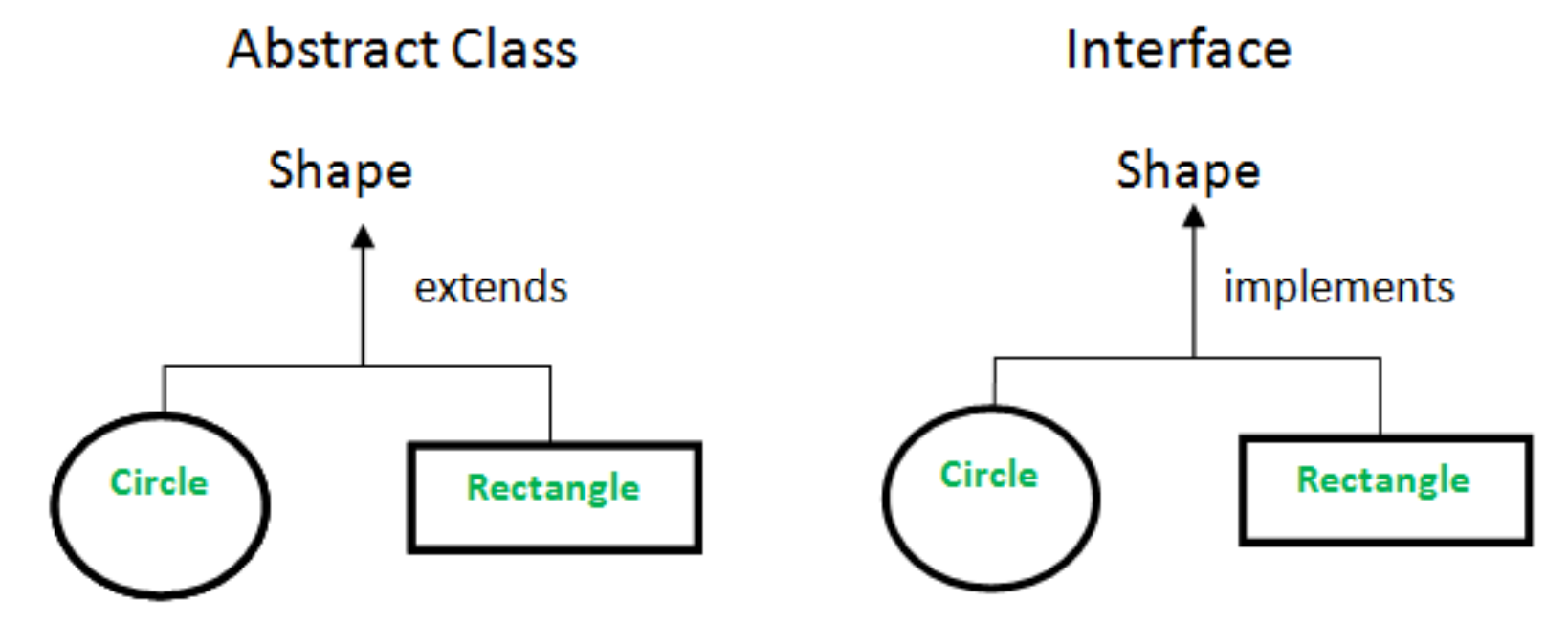
Interfaces

Another way to achieve abstraction in Java, is with interfaces.

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

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

INTERFACES


Abstract Class
Shape
extends
Circle
Rectangle
Interface
Shape
implements
Circle
Rectangle

INTERFACES

ההבדל בין מחלקה אבסטרקטית לממשק

- מחלקה יכולה להרחיב רק מחלקה מופשטת אחת אבל יכולה ליישם ממשקים רבים יחדיו.
- מחלקה אבסטרקטית מאפשרת לך ליצור מתודות שתתי-מחלקות יכולות ליישם או לעקוף ואילו ממשק רק מאפשר לך לציין מתודות אך לא מיישם אותם.

- **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
- **Type of variables:** Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.