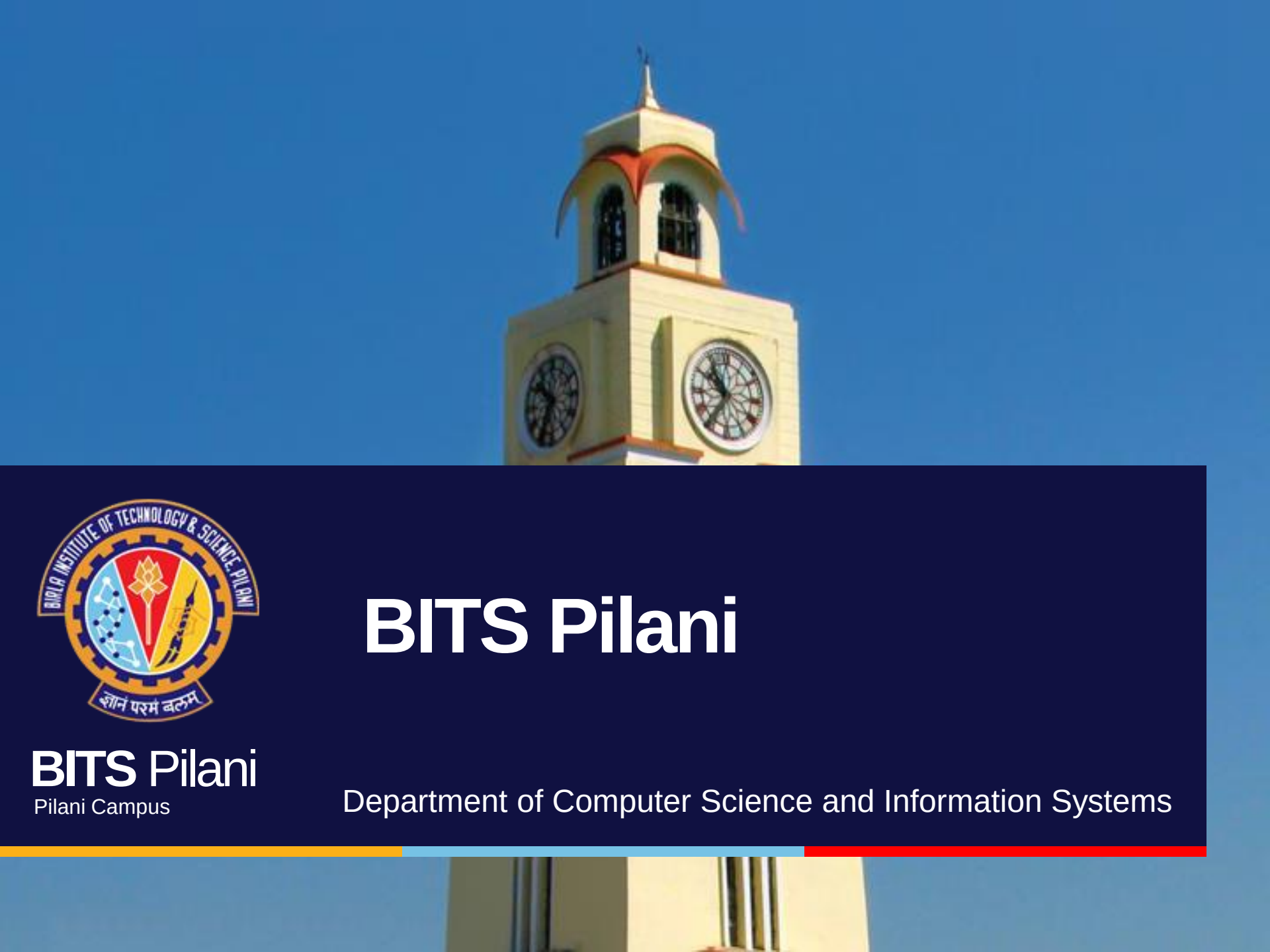


# BITS Pilani

**BITS** Pilani
Pilani Campus

Department of Computer Science and Information Systems


# Object Oriented Programming (CS F213)

**Tanmaya Mahapatra**
**Tanmaya.mahapatra@pilani.bits-pilani.ac.in**
**Chamber# 6121-Y, NAB**

**Consultation Hour**
**By appointment through mail**

# Introduction to Object Oriented Programming



- **Object-Oriented Programming (OOP)** is the term used to describe a programming approach based on **objects** and **classes**.
- Allows us to organize software as a collection of objects that consist of both data and behavior.
- Conventional functional programming practice only loosely connects data and behavior.
- Object-oriented programming is the most important and powerful way of creating software.
- The object-oriented programming approach encourages:
  - Modularization: where the application can be decomposed into modules.
  - Software re-use: where an application can be composed from existing and new modules.

# Features of OOP

- An object-oriented programming language generally supports five main features:
  - Classes
  - Objects
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Abstraction

# An Object-Oriented Class

- If we think of a real-world object, such as a television it will have several features and properties:
  - We do not have to open the case to use it.
  - We have some controls to use it (buttons on the box, or a remote control).
  - We can still understand the concept of a television, even if it is connected to a DVD player.
  - It is complete when we purchase it, with any external requirements well documented.
  - The TV will not crash!

In many ways this compares very well to the notion of a class.

# An Object-Oriented Class

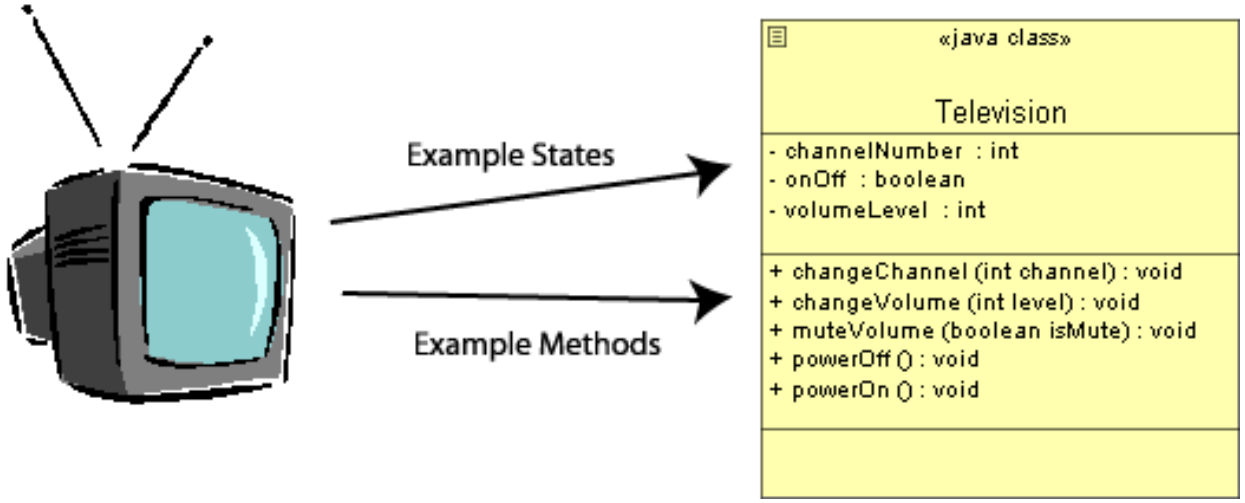


- A class should:
  - Provide a well-defined interface - such as the remote control of the television.
  - Represent a clear concept - such as the concept of a television.
  - Be complete and well-documented - the television should have a plug and should have a manual that documents all features.
  - The code should be robust - it should not crash, like the television.

- With a functional programming language (like C)
  - The component parts of the television scattered everywhere.
  - Users would be responsible for making them work correctly.
  - No case surrounding the electronic components.

# An Object-Oriented Class

- Humans use class based descriptions all the time (Think about this, we will discuss it soon.)
- Classes allow us a way to represent complex structures within a programming language. They have two components:
  - States - (or data) are the values that the object has.
  - Methods (or behavior) - are the ways in which the object can interact with its data, the actions.

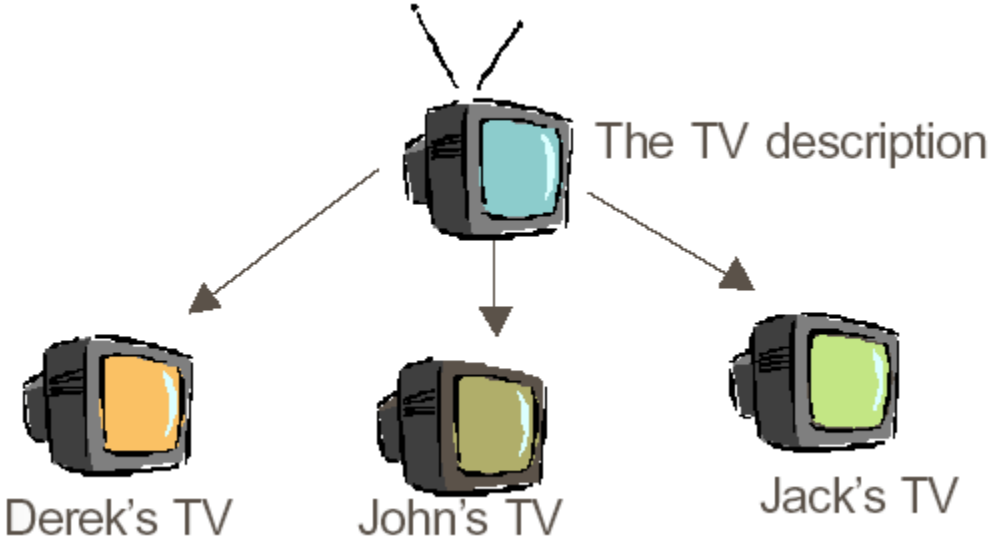



**Unified Modelling Language (UML) representation of the Television class for object-oriented modelling and programming.**

# An Object

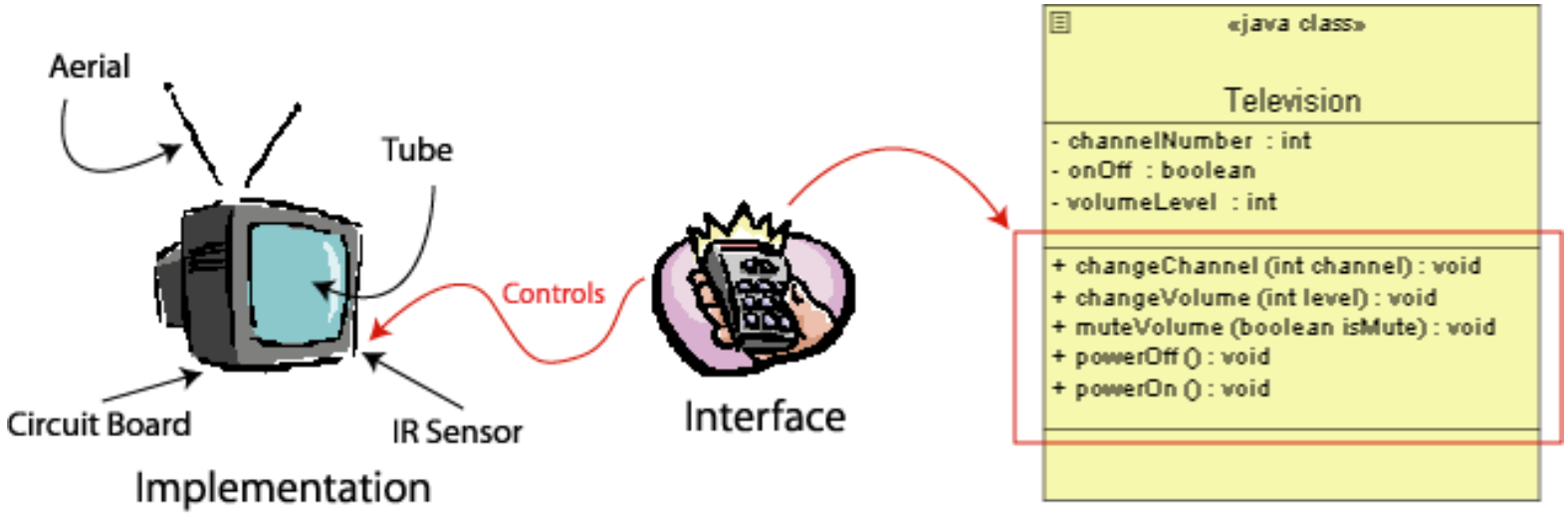
- An object is an instance of a class.
- Class is a description/blueprint of a concept
- An object is a realization of this description/blueprint to create an independent distinguishable entity.
- For example, in the case of the Television, the class is the set of plans (or blueprints) for a generic television, whereas a television object is the realization of these plans into a real-world physical television. So there would be one set of plans (the class), but there could be thousands of real-world televisions (objects).
- Objects have their own identity and are independent from each other.
  - If the channel is changed on one television it will not change on the other televisions.
- Objects can be concrete (a real-world object, a file on a computer) or could be conceptual (such as a database structure) each with its own individual identity.

# An Object



# Encapsulation

- Encapsulation is used to hide the mechanics of the object, allowing the actual implementation of the object to be hidden. All we need to understand is the interface that is provided for us.
  - For example, Television class, where the functionality of the television is hidden from us, but we are provided with a remote control, or set of controls for interacting with the television, providing a high level of abstraction.




- There is no requirement to understand how the signal is decoded from the aerial and converted into a picture to be displayed on the screen before you can use the television.

# Encapsulation

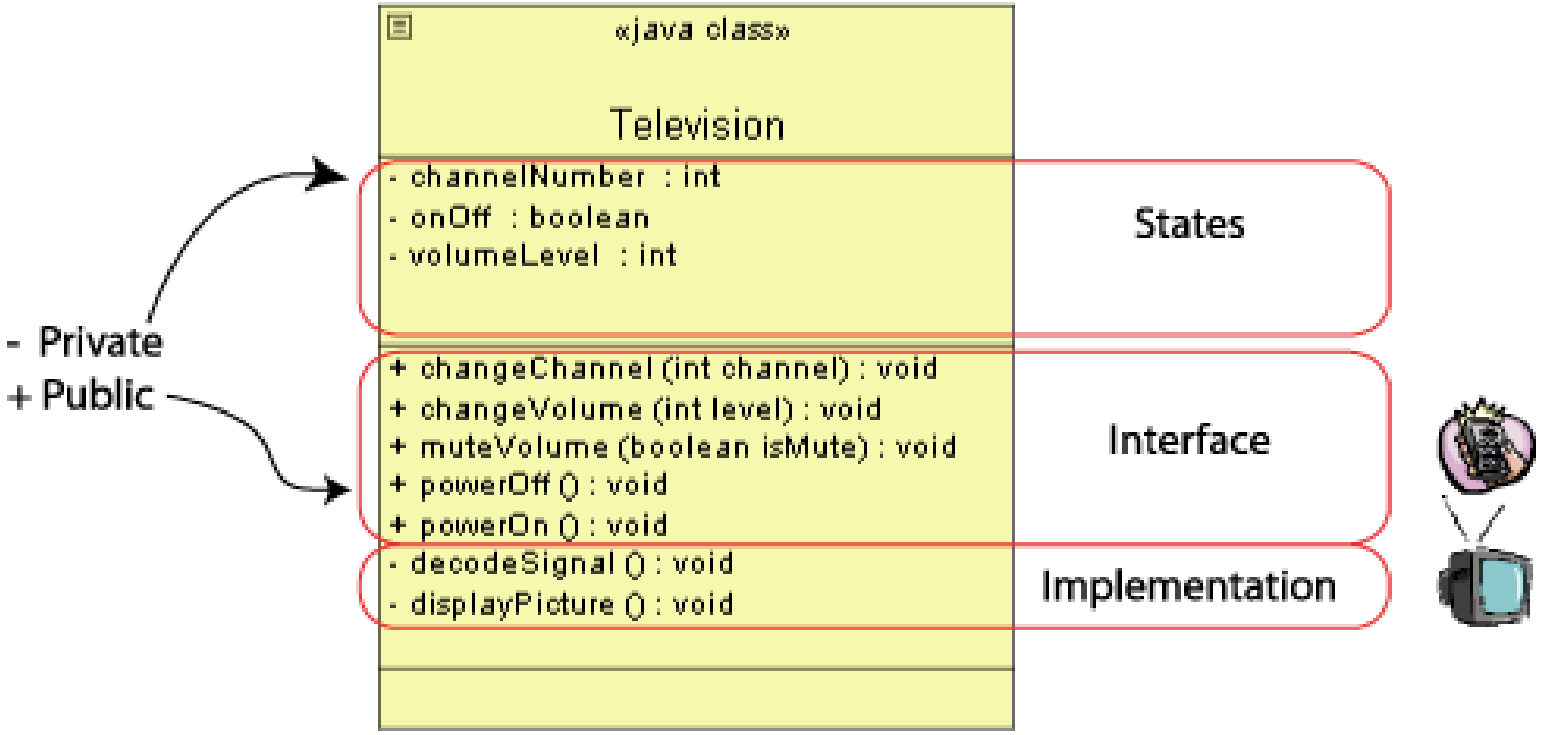
- There is a sub-set of functionality that the user is allowed to call, termed the interface.
  - What are the interfaces of a Car?
- The full implementation of a class is the sum of the public interface plus the private implementation.
- Encapsulation is the term used to describe the way that the interface is separated from the implementation.
- Encapsulation as data-hiding
  - Allow certain parts of an object to be visible
  - Advantages for both the user and the programmer
  - User
    - Need only understand the interface.
    - Need not understand how the implementation works or was created.
  - Programmer
    - Can change the implementation, but need not notify the user.

# Encapsulation

- There is a sub-set of functionality that the user is allowed to call, termed the interface.
  - What are the interfaces of a Car?
- The full implementation of a class is the sum of the public interface plus the private implementation.
- Encapsulation is the term used to describe the way that the interface is separated from the implementation.
- Encapsulation as data-hiding
  - Allow certain parts of an object to be visible
  - Advantages for both the user and the programmer
  - User
    - Need only understand the interface.
    - Need not understand how the implementation works or was created.
  - Programmer
    - Can change the implementation, but need not notify the user.

# Encapsulation

- We can identify a level of 'hiding' of particular methods or states within a class using the public, private and protected keywords:
  - public methods - describe the interface.
  - private methods - describe the implementation.

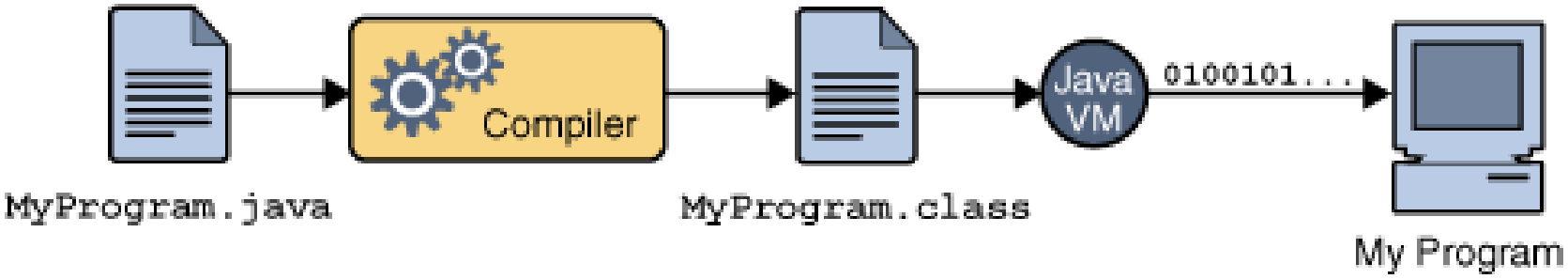


# Java™ Technology

- Characterized by the following keywords
  - Simple
  - Architecture Neutral
  - Objected Oriented
  - Portable
  - Distributed
  - Multithreaded
  - High Performance
  - Robust
  - Dynamic
  - Secure

- Each of these keywords are very well explained in ***"Java Language Environment"***, a white paper written by James Gosling and Henry McGilton

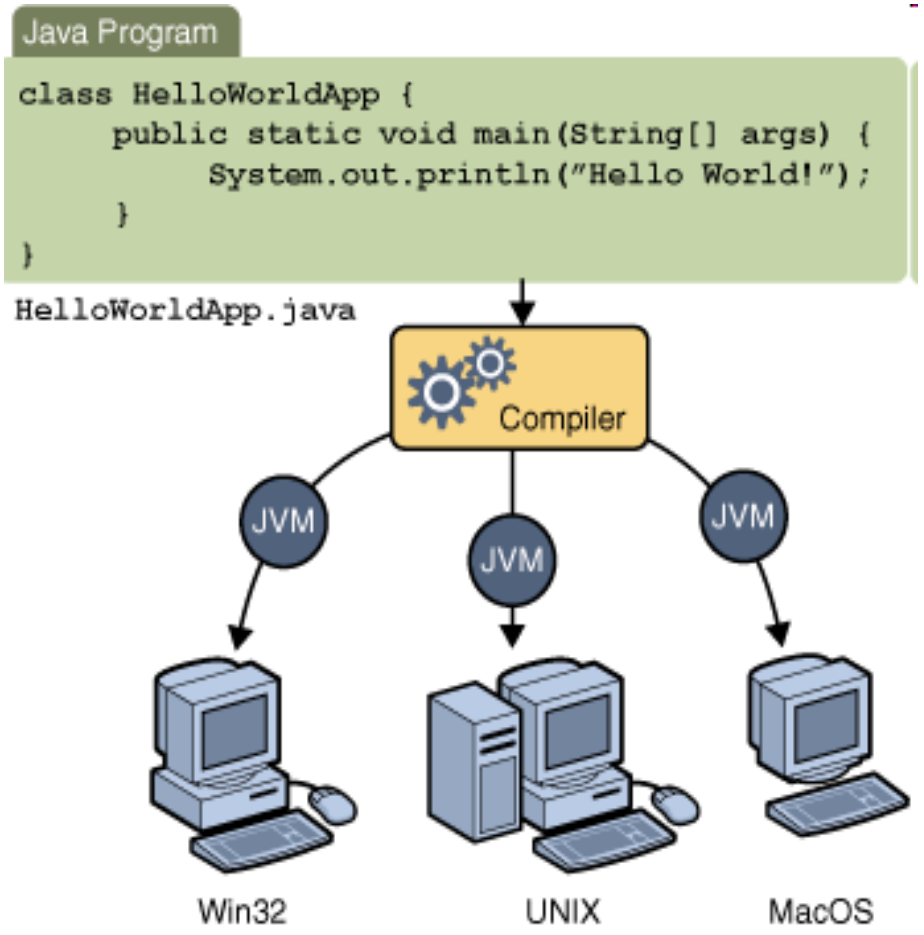
# The Java™ Programming Language




- All source code is first written in plain text files ending with the .java extension
- Those source files are then compiled into .class files by the javac compiler
- A .class file does not contain code that is native to your processor; it instead contains ***bytecodes*** — the machine language of the Java Virtual Machine (Java VM)
- The java launcher tool then runs your application with an instance of the Java Virtual Machine

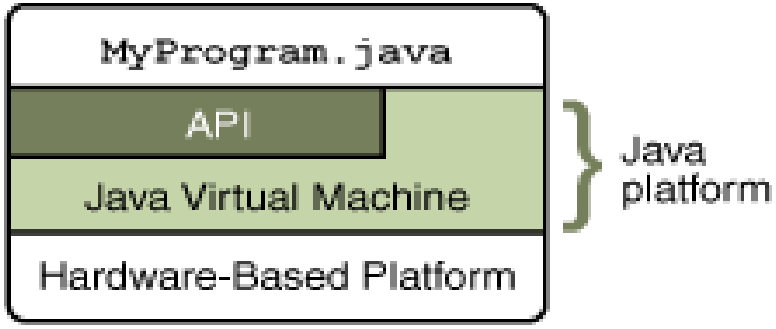
# The Java™ Programming Language

- Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris TM Operating System (Solaris OS), Linux, or Mac OS
- Through the Java VM, the same application is capable of running on multiple platforms.



# The Java Platform

- A *platform* is the hardware or software environment in which a program runs
- Microsoft Windows, Linux, Solaris OS, and Mac OS
- Most platforms can be described as a combination of the operating system and underlying hardware
- The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.




- The Java platform has two components:
  - **The *Java Virtual Machine***
  - **The *Java Application Programming Interface* (API)**
- The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as ***packages***

# Creating Your First Application

innovate achieve lead

- Checklist
  - ***The Java SE Development Kit***
  - ***A text editor***

- Creating application using simple text editor
  - ***Install JDK***
  - ***Check for correct installation***
  - ***Create HelloWorldApp.java***
  - ***Compile HelloWorldApp.java***
  - ***Run HelloWorldApp***

- Creating application using Eclipse

# A Closer Look at HelloWorldApp



```
The HelloWorldApp class implements an application that
simply prints "Hello World!" to standard output.
```

```
class HelloWorldApp {
    public static void main(String[] args) {
        // Display the string.
        System.out.println("Hello World!");
    }
}
```

The HelloWorldApp class definition

The most basic form of a class definition is

```
class name {
 . . .
}
```

- Every application in Java should have a main() method
- modifiers public and static can be written in either order
- It is the entry point of your application
- Accepts a single argument which is an array of elements of type String
- Mechanism through which the runtime system passes information to your application

# A Sample Java Program



**Display 1.1 A Sample Java Program**

Name of class (program)

The `main` method

```
1   public class FirstProgram
2   {
3       public static void main(String[] args)
4       {
5           System.out.println("Hello reader.");
6           System.out.println("Welcome to Java.");

7           System.out.println("Let's demonstrate a simple calculation.");
8           int answer;
9           answer = 2 + 2;
10          System.out.println("2 plus 2 is " + answer);
11      }
12  }
```

**SAMPLE DIALOGUE 1**

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

# System.out.println



- Java programs work by having things called *objects* perform actions
  - **`System.out`**: an object used for sending output to the screen
- The actions performed by an object are called *methods*
  - **`println`**: the method or action that the **`System.out`** object performs

# System.out.println



- *Invoking* or *calling* a method: When an object performs an action using a method
  - Also called *sending a message* to the object
  - Method invocation syntax (in order): an object, a dot (period), the method name, and a pair of parentheses
  - Arguments: Zero or more pieces of information needed by the method that are placed inside the parentheses

```
System.out.println("This is an argument");
```