

# Chapter One

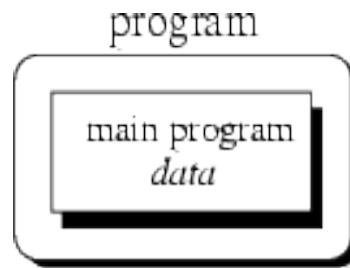
## Introduction

# Programming paradigm

- A computer program usually consists of two elements:
  - Data – characteristics
  - Code – action
- **A programming paradigm**
  - A fundamental style of programming
  - Provides the view that the programmer has about the execution of the program
- **Types**
  - Unstructured Programming
  - Procedural programming
  - Structured Programming
  - Object Oriented Programming

# Unstructured Programming

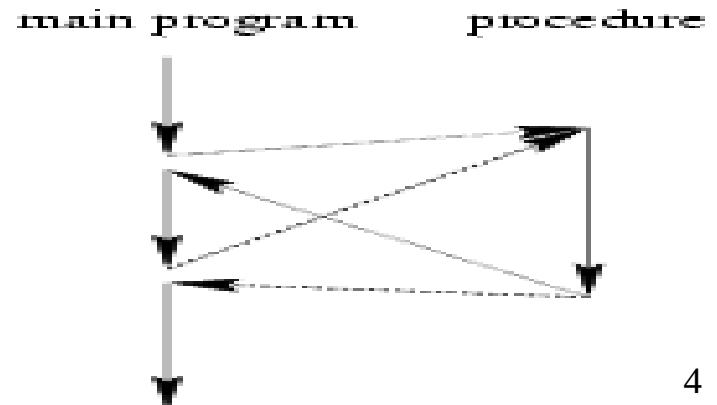
- consisting only of **one large** (usually main) program
- “main program” stands for a sequence of commands or *statements*
  - data is *global* throughout the whole program



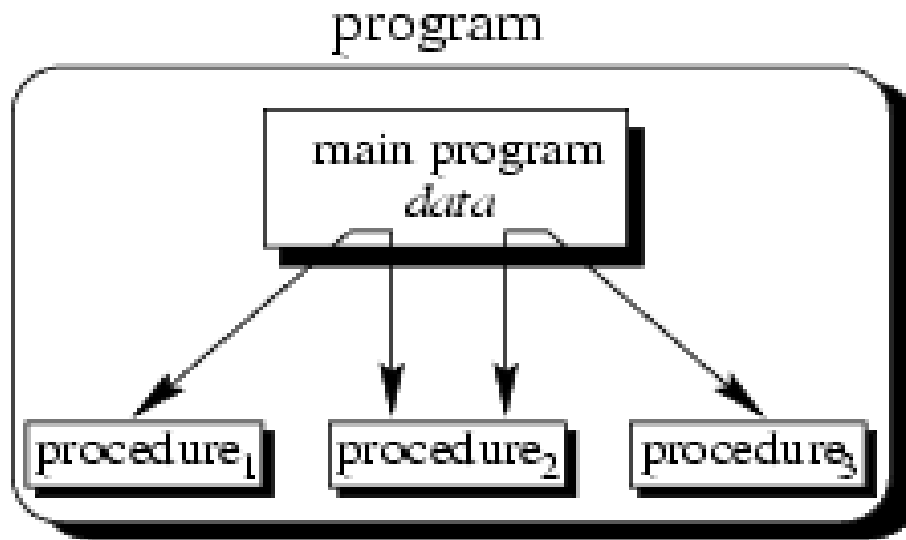
- **disadvantages**
  - Complex
  - The same statement sequence must be copied more than once in the program
- Ex. COBOL, FOCAL, MUMPS

# Procedural programming

- based upon the concept of *procedure call*
- A ***procedure call*** is used to invoke the procedure
- **Procedures** (routines, subroutines, methods, functions) simply contain a series of computational steps to be carried out to solve a problem
- Is a better choice than unstructured programming
- Involves moderate complexity or require significant ease of maintainability
- The same code can be reused without copying it
- Is strongly modular or structured



- We have a single program, which is divided into small pieces called **procedures**



- **Advantage**

- to re-use the same code at different places in the program without copying it
- easier way to keep track of program flow

- **Example**

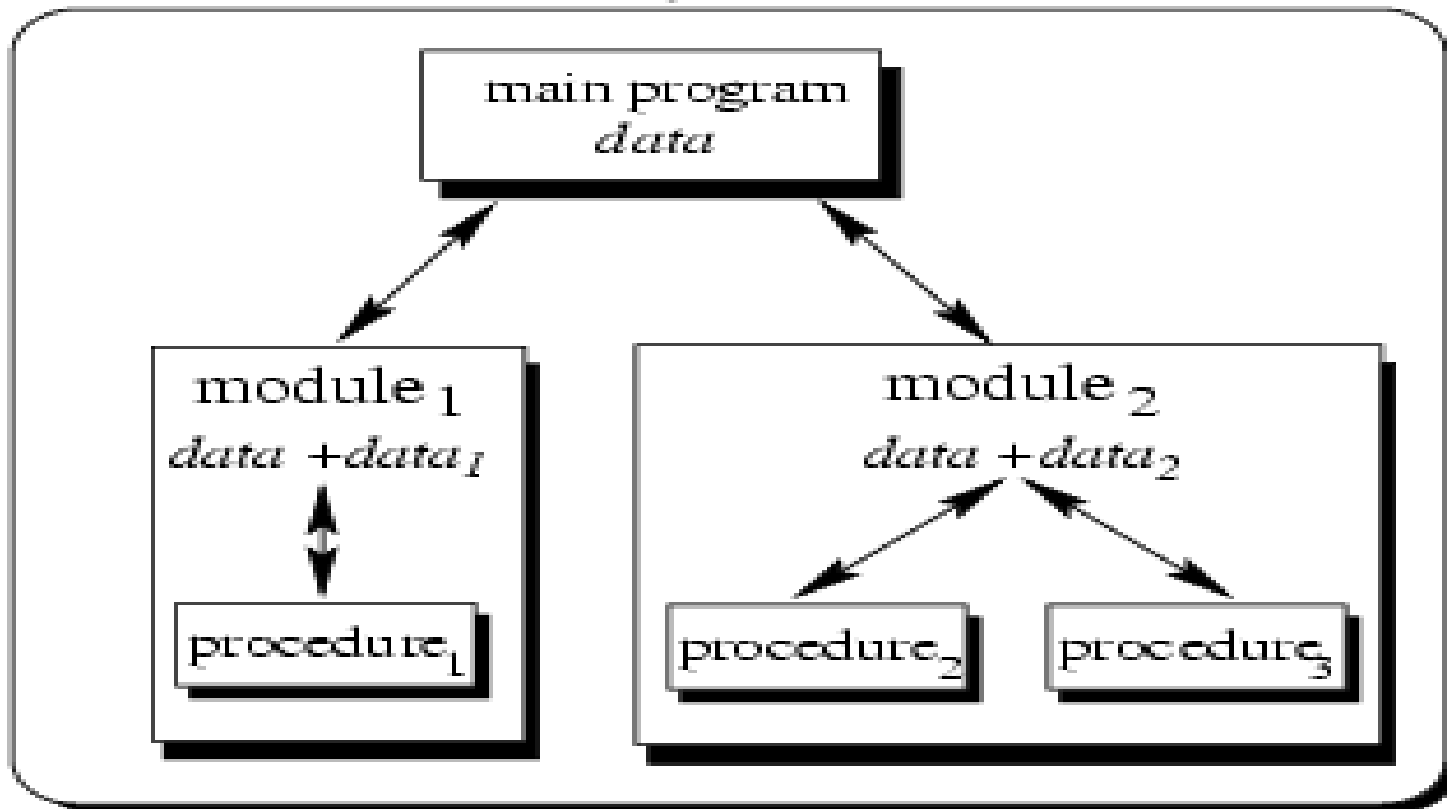
- FORTRAN, ADA

# Structured Programming

- is a subset of procedural programming (also known as *modular programming*)
- **Enforces a logical structure** on the program being written to make it more efficient and easier to understand, and modify.
- A program is divided into several smaller parts called **module** which interacts through procedure calls
- Each module can have its own data
  - allows each module to manage an internal *state* which is modified by calls to procedures of this module
- Employees top-down design model
  - map out the overall program structure into separate subsections
- Example
  - PASCAL, C

- **Advantage**

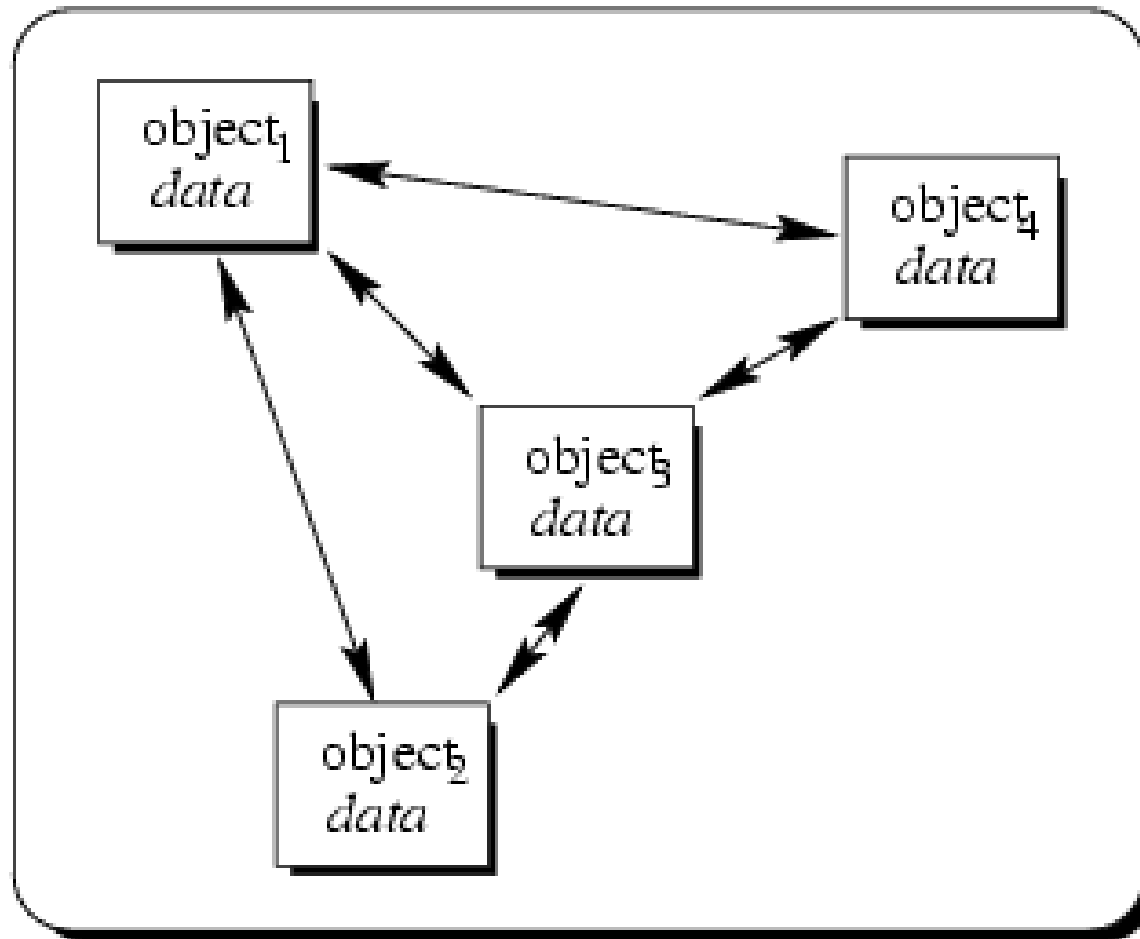
- Reuse
- easier to understand and modify  
program



# Object Oriented Programming - OOP

- **Data and operations** are grouped together
- Each object is capable of
  - receiving messages,
  - processing data, and
  - sending messages to other objects
- Uses objects and their interaction to design applications and computer programs
- A program is a collection of individual units, or **objects**
- Organizes programs around data – Object.
- write programs by modelling problem as set of collaborating components
  - you determine what the building blocks are
  - then put them together so they cooperate properly

program

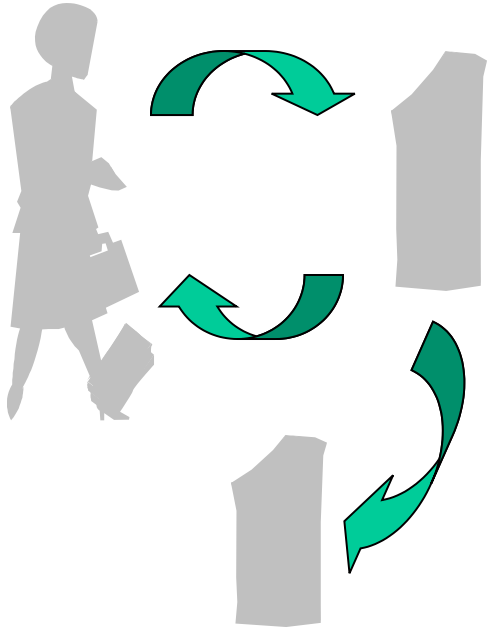


# Procedural vs. Object-Oriented

- how to write the logic, not how to define the data
- unit programming is function
- separates the data of the program from the operations
- we want to manipulate objects rather than the logic required to manipulate them
- unit in object-oriented programming is objects
- focus on both the data and the operation

# Procedural vs. Object-Oriented

- Procedural



Withdraw, deposit, transfer

- Object Oriented



Customer, money, account

# The Big OO Concepts

- It is a technique of modeling some kind of system based on objects
- Basic OO concepts
  - Models
  - Objects
  - Classes
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

# Modeling

- Successful programs solve problems in the real world.
  - They correspond closely to the real world problems they solve.
  - They model the problem domain and users' activities.
- Modeling enables better communication and visualization *for all stakeholders*.
- Successful OO designs almost always begin with a visual “object model” of the problem domain, involving both the **domain experts and software designers**.
- Would you let a contractor build your new house without blueprints?
- The essence of modeling is to show all *pertinent* detail.

# Object

- Represents a real world entity
- possesses operations and attributes (data)
- Possesses all the know-how and information it needs to perform the services for which it was designed
  - E.g. Building, Employee, Patient etc.
- Consists of **attributes** (data) and **behaviors** (methods)
  - The interface of the object is the attributes and methods that are visible (available) from the outside of the object for other objects
  - Is self-consistent, coherent, and complete.
  - Is (usually) not very complex or large.
  - Is as *loosely coupled* with other objects as possible.

# Characteristics of Objects

- They have unique identity.
- They fall into categories, or classes.
- They fall into hierarchies or aggregations.
- They have well defined behaviors & responsibilities.
- They separate interface from implementation.
- They hide their internal structures.
- They have states.
- They provide services.
- They send messages to other objects.
- They receive messages from other objects, and react appropriately.
- They delegate responsibilities to other objects.

# Object Examples

- **Example 1: Dogs**
  - **States:** name, color, breed, and “is hungry?”
  - **Behaviors:** bark, run, and wag tail
- **Example 2: Cars**
  - **States:** color, model, speed, direction
  - **Behaviors:** accelerate, turn, change gears

# Class

- A class is a **template (blueprint)**
- used to create multiple objects with similar features
- Is an abstract description of the data (characteristic) and behavior of a collection of similar objects.
- A class can have: **attributes** and **behaviors**
  - Attributes (***fields***): data variables which determine the status of the class or an object
  - behavior (***methods***): executable code of the class built from statements.
    - It allows us to manipulate/change the status of an object or access the value of the data member
- Classes are to objects as blueprints are to houses
- The classification(s) of a collection of objects often depends on the attributes in which you are interested.
  - **Examples:** streets, roads, and highways...

- Classes can be considered as data type
- The properties and methods defined by a class are called **members**
- Even though all dogs have four legs, and bark, each dog's behavior is independent of other dogs.
  - For example: Dog #1 is a black Poodle, Dog #2 is a red Irish Setter

# Object vs. Class

- “Class” refers to a blueprint. It defines the variables and methods the objects support
- “Object” is an instance of a class. Each object has a class which defines its data and behavior.
  - Example : Class – Girl
    - Object- Abebech, Lili, Kiki ...
- The world conceptually consists of objects
- We call the object type a class
- An Object is instantiated from a Class
  - Example

**BankAccount myAccount ;**

**myAccount = new BankAccount**

# Encapsulation

- The only thing that an object knows about another object is the object's interface
- Exposing only the details that are relevant: the *public interface*.
- *What?* not *How?*
- Hiding the “gears and levers”.
- Exposing only the client's view of the object's responsibilities
- Protects the object from outside interference.
- Protects other objects from relying on details that are likely to change.
- It allows the developer to separate an object's implementation from its behavior.

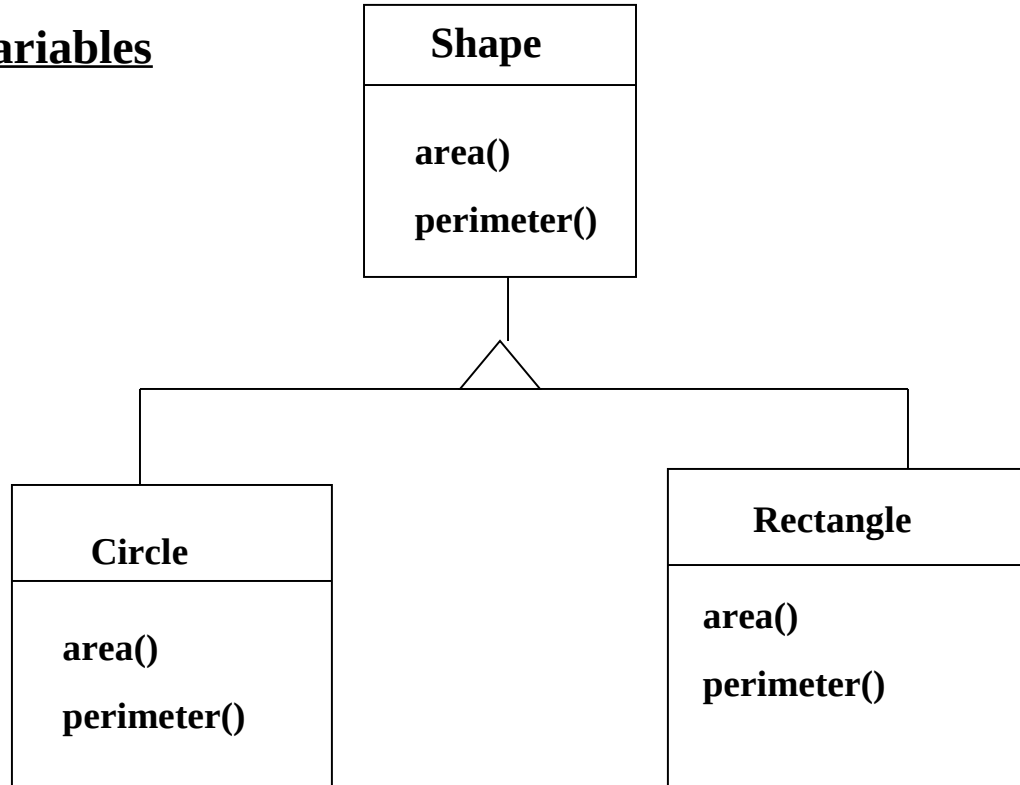
- As long as the interface remains the same, any changes to the internal implementation are transparent to the user
- *Information hiding* promotes *loose coupling* between objects and modularity, which promotes flexible design, which promotes reusability.
- Reduces interdependencies in the code.
  - **Example:** Your car's gas pedal.
- **Best practice:** Objects only speak to each other by method (i.e. function) calls. They never directly access each other's attributes.

# Polymorphism

- “The ability of two or more classes of objects to respond to the same message, each in its own way.”
- “The ability to use any object which implements a given interface”
- works together with encapsulation and inheritance to simplify the flow control in an object-oriented program
- allows a sending object to communicate with different objects in a consistent manner without worrying about how many different implementations of a message.
- The sending object does not have to know how the receiving object implements the message. Only the receiving objects worries about that

# Polymorphism

## Polymorphic Variables



- Rectangle and Circle are concrete classes with their own separate implementations of the methods **area()** and **perimeter()**

# Inheritance

- allows a class to have the same behavior as another class and extend or tailor that behavior to provide special action for specific needs.
  - **Example:** “Class Y is like Class X except for the following differences...”
- Classes that inherit from a class are called *subclasses (Derived Class)*.
- The class a subclass inherits from are called *superclass (Base Class)*.
- Why use inheritance?
  - When you have two types where one is necessarily an extension of the other.
  - Sometimes (but not all the time) you are going to want to ignore the differences and look only at the what they have in common (the base class).

- This is called generalization.
- Consider a system that can manipulate various kinds of shapes:
- The sub-class inherits the base class' data members and member functions
- A sub-class has all data members of its base-class plus its own
  - *Base class = parent class = superclass.*
  - *Derived class = child class = subclass.*
- is-a relationship: **inheritance**
  - A Square is-a-kind-of Rectangle (uses inheritance).
- has-a relationship: **composition**
  - A clock has a date

- **Abstraction**

- is simplifying complex reality by modeling classes appropriate to the problem, and
- working at the most appropriate level of inheritance for a given aspect of the problem.
- Is achieved through **Composition**

# Summary of OO Concepts

- Everything is an object.
- Computation is performed by objects communicating with each other, requesting that other objects perform actions.
- Each object has its own memory.
- Every object is an instance of a class.
- The class is the repository (storage) for behavior associated with an object.
- Classes can be organized into a single rooted tree structure called inheritance hierarchy

# History and Evolution

- **What is Java?**

- Java is an **object-oriented** programming language developed by **Sun Microsystems**
- It is intended for the development of platform independent software
- the Java language was designed to be small, simple, and portable across platforms

- **History and Evolution**

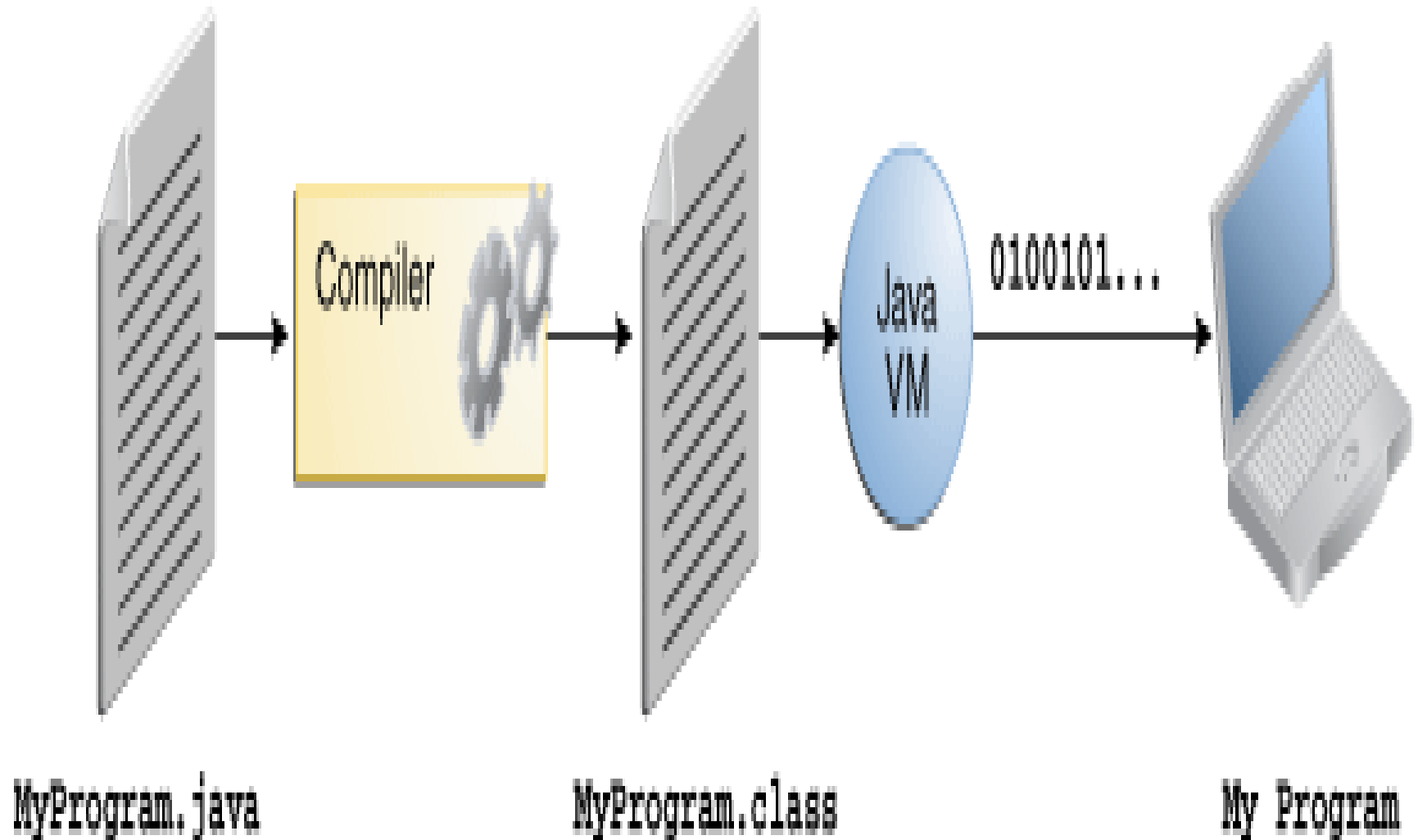
- The Java language was developed at Sun Microsystems in 1991
- The goals were to be small, fast, efficient, and easily portable to a wide range of hardware devices.

- **James Gosling** and his team developed a new language for **Star7**.
- The language was based on **C and C++**
- It was originally intended for writing programs that control consumer appliances such as toasters, microwave, ovens, and others.
- The language was first called **Oak**, named after the oak tree outside of Gosling's office,
- but the name was already taken, so the team renamed it **Java**.
- In 1993, the Green team added **applet programming**
- In 1994, the team developed a web browser called “**Hot Java**” to locate and run applet programs on the Internet.
- In early 1995, it is launched as an **Internet extension** for browser.

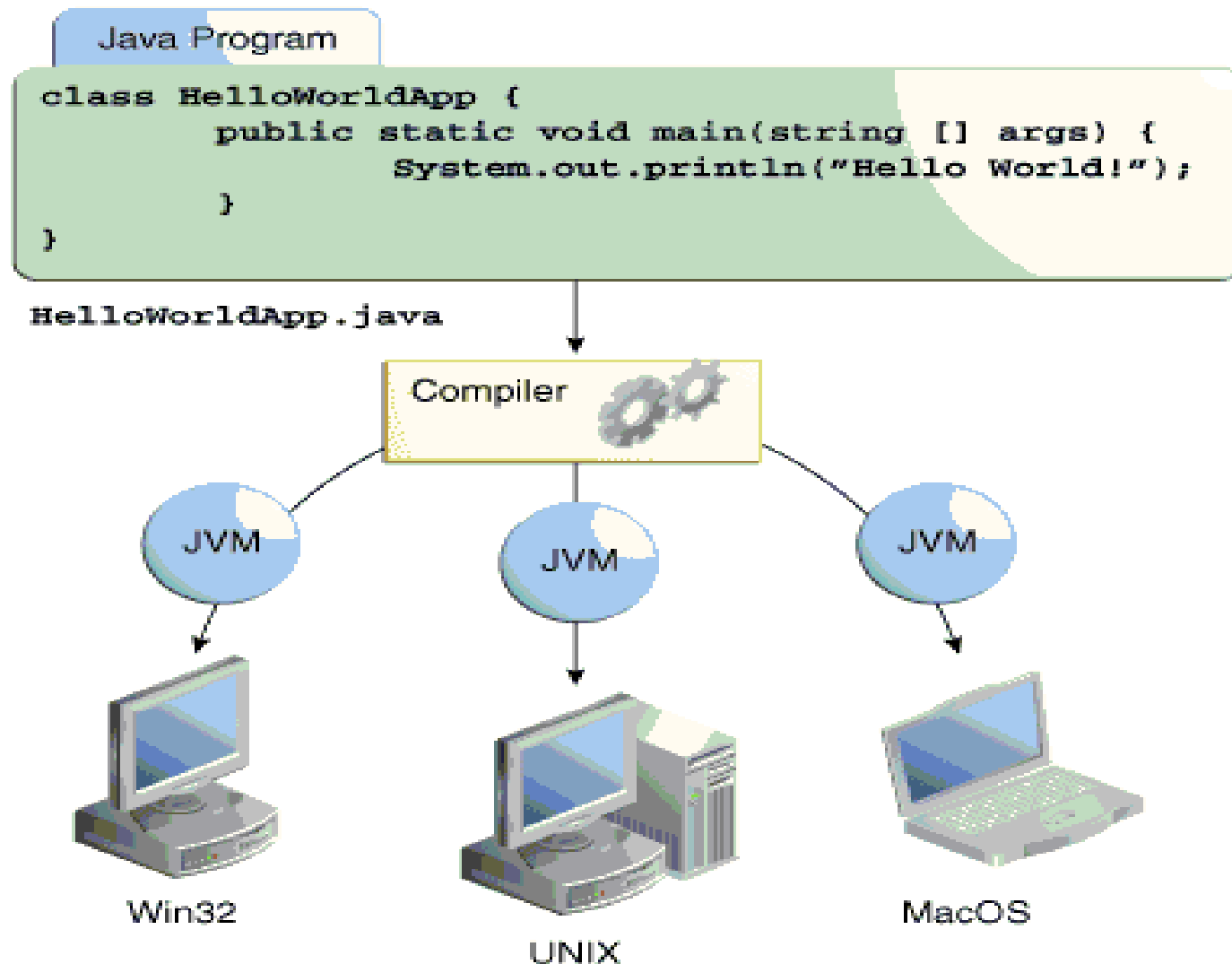
# Java Technology

- Java technology is both a programming language and a platform.
- **The Java Programming Language**
  - The Java programming language is a high-level language
  - all source code is first written in plain text files ending with the **.java** extension
  - 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**<sup>1</sup> (Java VM)
  - The **java** launcher tool then runs your application with an instance of the Java Virtual Machine.

# An overview of the software development process.



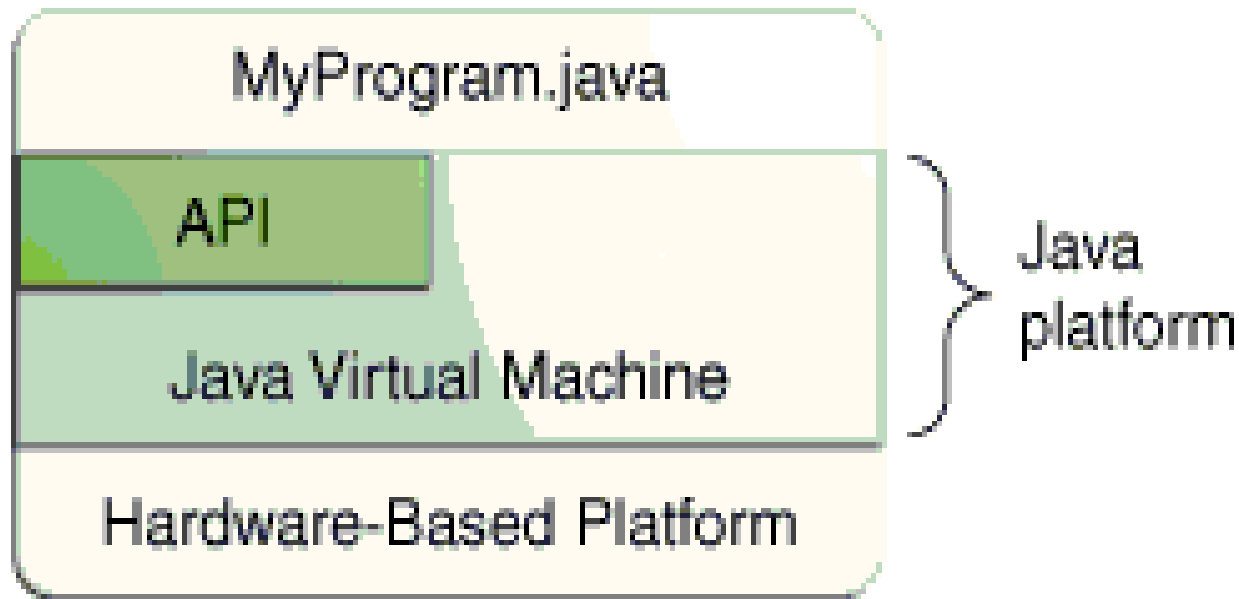
- Through the Java VM, the same application is capable of running on multiple platforms.



- **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)*
- JVM is the base for the Java platform and is ported onto various hardware-based platforms.
- The API is a large collection of ready-made software components that provide many useful capabilities.
- API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*

- The API and Java Virtual Machine insulate the program from the underlying hardware.



- As a platform-independent environment, the Java platform can be a bit slower than native code.

# Java Language Characteristics

- **Simplicity**

- Initial design goals was to be small and simple,
  - easier to write, easier to compile, easier to debug, and, best of all, easy to learn
- Syntax and object-oriented structure is borrowed from C++
- More complex parts of those languages have been excluded
  - does not support the struct, union, and pointer data types.
  - does not support typedef or #define.
  - does not support multiple inheritance.
  - handles command-line arguments differently than C or C++.
  - has a String class. This differs from the null-terminated array of characters
  - There are no pointers in Java, nor is there pointer arithmetic

- **Automatic Memory Management**
  - Java takes care of referencing and dereferencing memory objects
  - An automatic garbage collection mechanism manages allocation and freeing of memory
  - Array indexes, object types, and casting between different objects are always validated at compile-time and run-time.
- **Object-Oriented**
  - Structured in terms of *classes*, which group data with operations on that data
  - Can construct new classes by *extending* existing ones
  - Interfaces describe the functionality of a class, independent from its implementation, leveraging reuse of software components

- **Secure**

- Three mechanisms for security

- The bytecode is verified by the VM ensuring that no illegal code is included.
    - The code could be signed, i.e. cryptographic algorithms are applied to ensure that the code originates from someone the user trusts.
    - Uses the so called "sandbox"-model:
      - The program runs in a memory area of the system, which is controlled by the virtual machine.
      - The program which is allowed to perform any operation within the sandbox, has limited access to any system resources outside.
      - Unauthorized code cannot read or write any data on the system.

- **Distributed**

- Java is specifically designed to work within a networked environment
- By using Java's URL (Uniform Resource Locator) class, an application can easily access a remote server.
- Classes also are provided for establishing socket-level connections.

- **Interpreted**

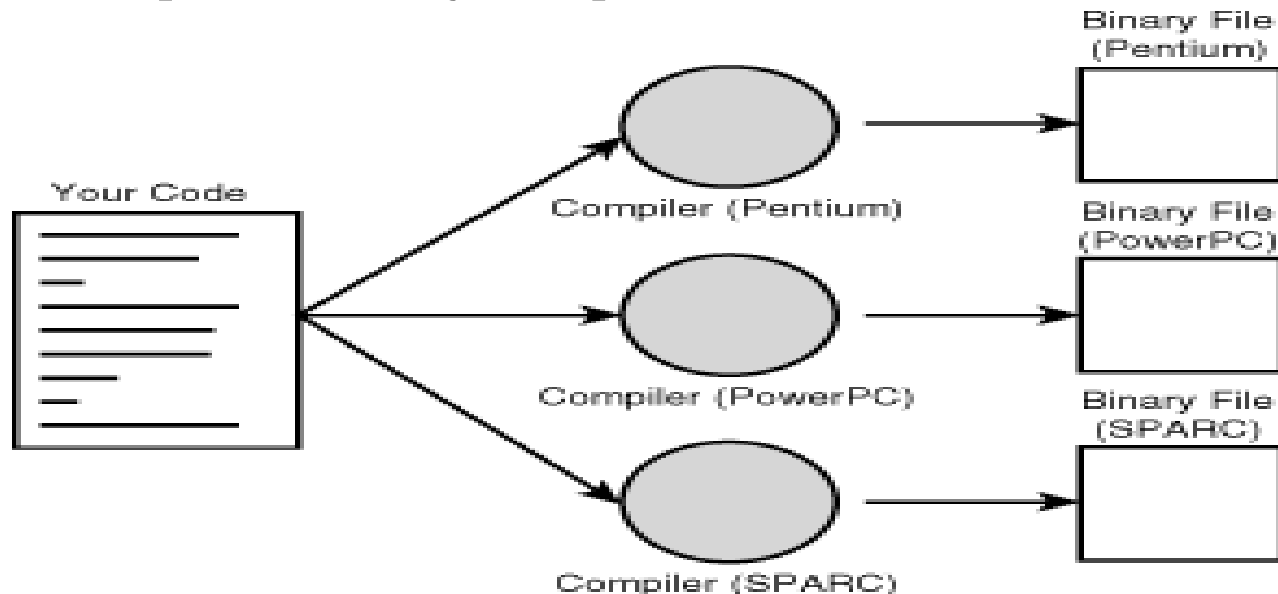
- the Java compiler translates a Java class source file to **bytecodes** which
  - can be run on any machine that runs a Java interpreter or Java-enabled browser
  - allows the Java code to be written independently of the users' platforms
  - eliminates the compile and run cycle for the client because it is not specific to a given machine but interpreted.

- **Robust**

- Robust software doesn't "break" easily because of programming bugs or logic errors in it.
- Strongly typed language
  - every variable must have a declared type.
  - Data type issues and problems are resolved at compile-time
  - all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility
  - Any type mismatches are errors that must be corrected before the compiler will finish compiling the class
- does not support direct pointer manipulation or arithmetic
  - makes it impossible for a Java program to overwrite memory or corrupt data
- uses runtime garbage collection instead of explicit freeing of memory

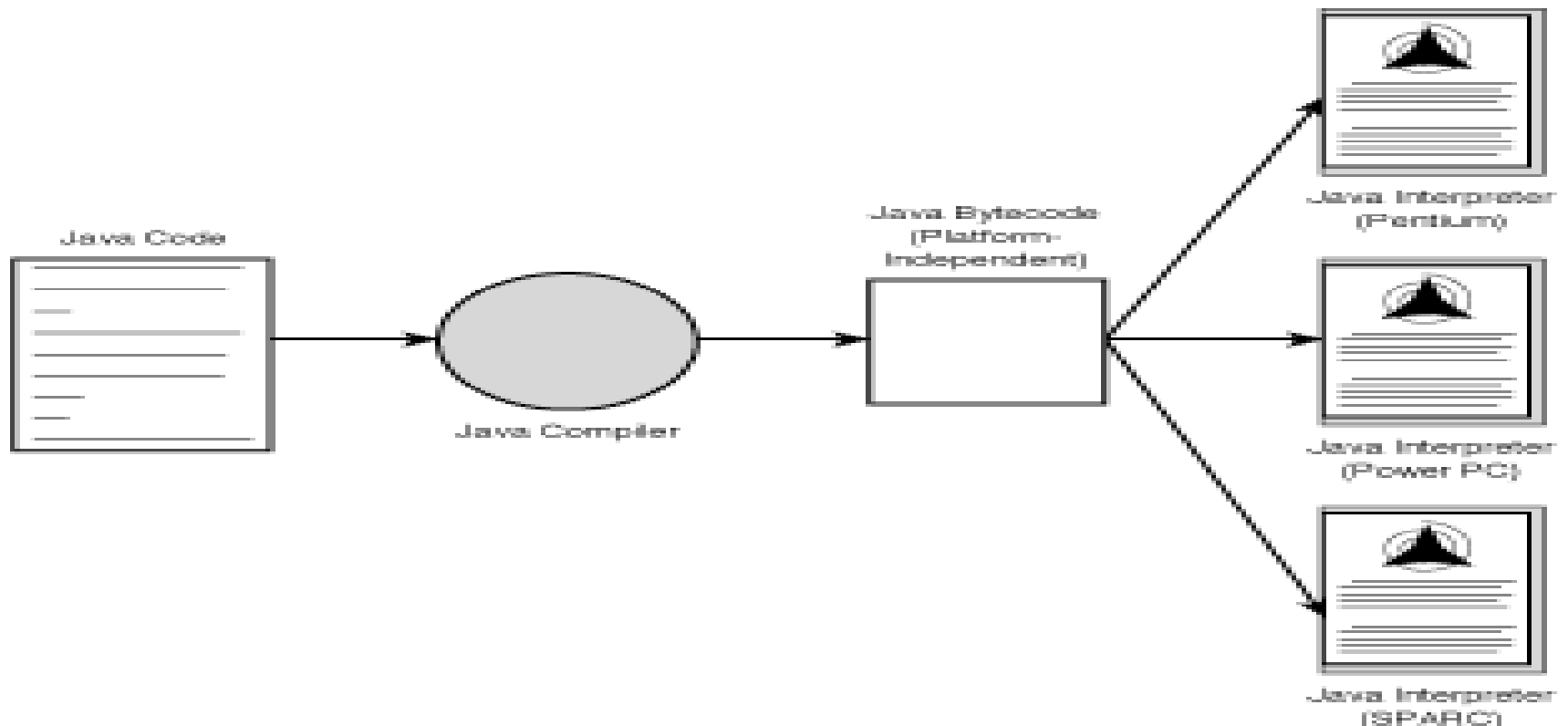
- **Architecture-Neutral**
  - the Java compiler creates bytecode instructions
  - interpreted by the Java interpreter
  - architecture neutrality is achieved in the implementation of the Java interpreter for each new architecture.
- **Portable**
  - the ability of a program to move easily from one computer system to another
  - **all primitive types** (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run

- The Java development environment has two parts:
  - a Java compiler
  - a Java interpreter
- **Java compiler**
  - takes your Java program
  - it generates bytecodes (instead of generating machine codes) from your source files
  - Bytecodes are instructions that look a lot like machine code, but are not specific to any one processor



## – **Java interpreter**

- reads the bytecodes
  - executes your Java program
- Java bytecode interpreter is often also called the Java virtual machine or the Java runtime.
- The disadvantage of using bytecodes is in execution speed



# Categories of Java programs

- ***Applet***
  - designed to be embedded in a Web page, and run by a browser
  - Applets run in a *sandbox* with numerous restrictions;
    - for example, they can't read files and then use the network
- ***Servlet***
  - designed to be run by a web server
- ***Application***
  - a conventional program

# JDK Editions

- Java is distributed as the JDK (Java Development Kit)
- instead of one all-purpose Java, different flavors of Java specifications:
  - The Standard,
  - Enterprise, and
  - Micro Edition
- **Java Standard Edition (J2SE)**
  - is the default language specification for workstations and small servers,
  - it is sufficient to install the Java 2 Runtime Environment (JRE)
  - The SDK includes all core packages of the language and the compiler required for creating Java byte code,
  - J2SE can be used to develop client-side standalone applications or applets.

- To execute software it is sufficient to just install the Java 2 Runtime Environment (JRE).
- Both, the JRE and the SDK run on Windows, various Unix systems, and Mac.
- All versions include a VM for executing Java byte code
- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages., JDBC (Java DataBase Connectivity), Java Message Service, JavaMail
  - Targets large server systems.

- **The Java 2 Micro Edition (J2ME)**
  - J2ME can be used to develop applications for mobile devices such as cell phones.
  - Introduced to meet the limitations and peculiarities of Pervasive Computing devices
  - is a platform on which very small and flexible Java application environments can be defined
  - includes:
    - virtual machines,
    - several libraries of APIs,
    - As well as tools for deployment and device configuration
  - Choosing subsets of the Java language particular for a specific device class is the basic concept of J2ME.
    - Each subset is called a **profile**
    - For example, the **Mobile Information Device Profile (MIDP)** defines the Java functionalities and APIs, which fit the screen size and memory constraints of PDA's

# Development Tools for Java

- The purpose of the JDK is to provide a complete set of tools for the development, testing, documentation, and execution of Java programs and applets
- Some of the programs in JDK are the following
  - ***javac*** - The Java language compiler, which converts your Java source code into Java Bytecode.
  - ***java*** - The Java language interpreter, which you may use to run Java application programs and applets.
  - ***javadoc*** – Creates an API documentation in HTML format from the Java source code
  - **Jdb**, The debugger, enables you to debug your Java classes
  - **appletviewer**, displays Java applets

- Many vendors provide tools to leverage Java development
  - Visual Café – Symantec
  - Jbuilder – Borland
  - Visual Studio J++ - Microsoft
  - Eclipse, JCreator, NetBeans, etc