ADVANCED OBJECT ORIENTED PROGRAMMING

Engr. Anees Ahmed Soomro

Assistant Professor

CS QUEST Nawabshah

https://anees-soomro.neocities.org

OOP CONCEPTS

1) Introduction of C++
2) C vs C++ vs JAVA
3) Compile and run first java program of **Hello Quest** message.
4) Compile and run of java program of **Check** class

Object Oriented Programming Language

- ***Both Java and C++ are most popular object-oriented programming languages***
- ***C++ was created at AT&T Bell Labs in 1979***
- ***Java was born in Sun Microsystems in 1990***

Language Feature Comparison

- ***Simple***
- ***Object-oriented***
- ***Distributed***
- ***Robust***
- ***Secure***
- ***Architecture Neutral***
- ***Portable***
- ***Compiled or Interpreted***
- ***High Performance***
- ***Multithreaded***
- ***Dynamic***
- ***Fun***

Simple

JAVA	C++
No pointer	*Pointer*
No multiple inheritance	*Multiple inheritance*
Automatic garbage collection	*Manual garbage collection*
No operator overloading	*Operator overloading*
No goto statement and no structure and union data structure	*Goto statement and structure and union data structure*

Purely Object-oriented

JAVA

- *No stand-alone data and functions*
- *Automatically supports polymorphism*

Hybrid Object-oriented

C++

- *Allows the stand-alone data and functions*
- *Needs declare virtual methods explicitly*

Distributed

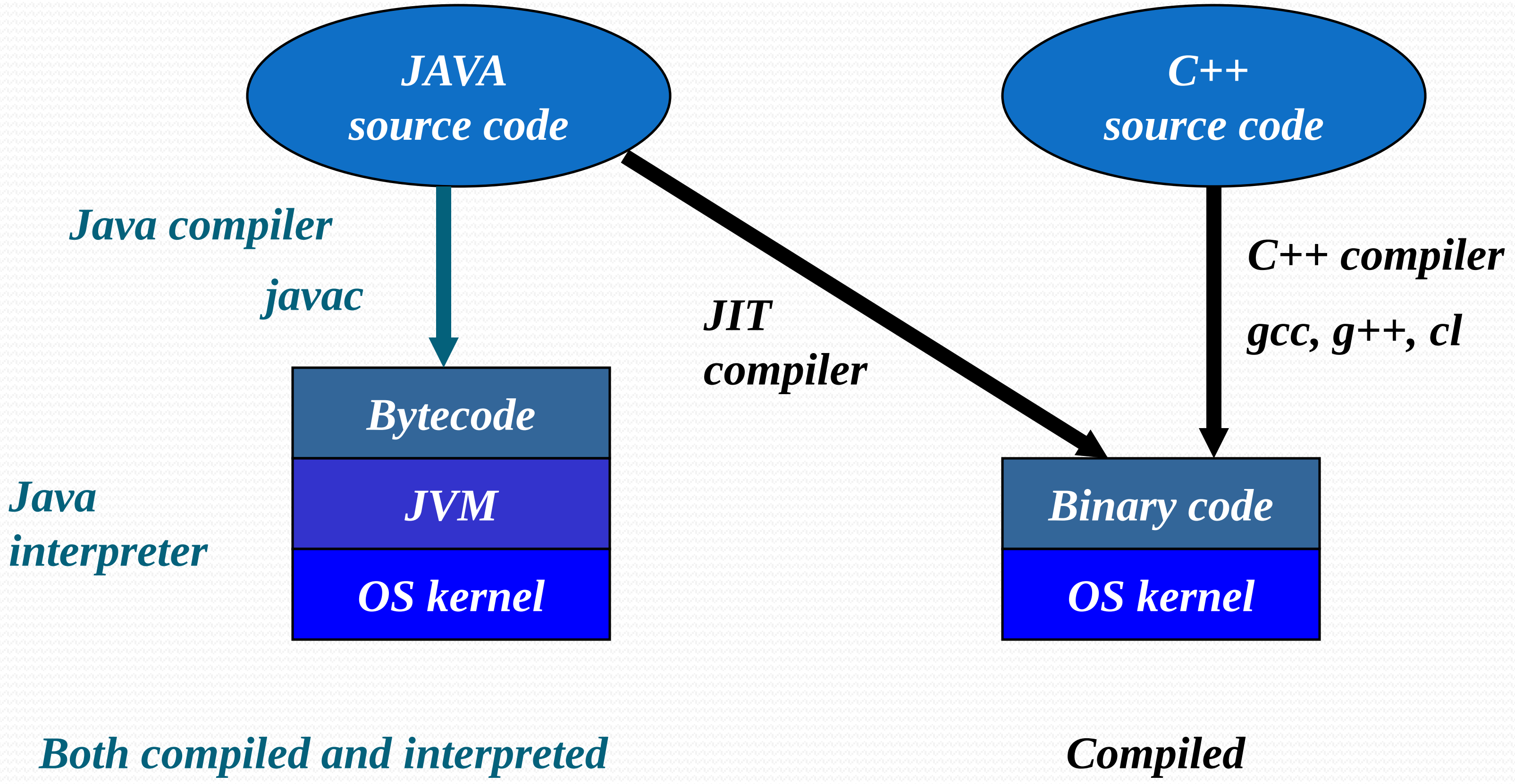
JAVA

Handles TCP/IP networking easily and nicely, can open and access objects across the Internet via URL just like a local file system

C++

External library supports TCP/IP networking, but much harder to do network programming

Interpreted or Compiled



High Performance

JAVA	C++
• *Much slower than C++, but good enough to run interactively for most applications*	• *About 10~20 times faster than equivalent JAVA code*
• *JIT compiler available*	• *Most operating systems are written using C/C++*

Robust

JAVA

- *Originally designed for writing highly reliable or robust software*
- *Explicit method declarations*
- *No pointers and automatic garbage collection avoid hard-to-debug mistakes*
- *Array bounds-checking*

C++

- *Allows implicit type and function declarations*
- *No automatic garbage collection is susceptible to memory leakage*
- *Using pointers is susceptible to memory corruption*
- *No array bounds-checking*

Secure

JAVA

- ***Byte-code is verified at run-time to ensure security restrictions are not violated***
- ***Memory layout is handled at run-time by JVM***
- ***Uses multiple namespaces to prevent hostile classes from spoofing a JAVA program***

C++

- ***Memory is handled at compile-time by compiler***

Architecture neutral and Portable

JAVA

- *Same Bytecode can run on any machine supporting JVM*

- *Well defined and fixed-size data types, file formats, and GUI behavior*

C++

- *Platform-dependent binary code cannot be executed on a different machine*

- *Implementation specific and varied-size data types by platforms*

Multithreaded

JAVA

- *Provides native multithreading support*
- *Concurrent applications are quite easy*

C++

- *Rely on external libraries for multithreading*
- *Harder to do multithreaded programming*

Dynamic

JAVA	C++
Run-time representation for classes makes it possible to dynamically link classes into a running system	*Needs recompile if libraries are updated*
Loads classes as needed, even from across networks	*Load libraries when compiled*

FUN FUN F U N

JAVA *Nice features combined with the Internet applications make JAVA programming appealing and fun*

C++ *The complicated or even some confusing features make C++ programming error prone*

Summary and Conclusion

- *C++ is a high performance and powerful language. Most of the industry* **software** *is written in C/C++*
- *JAVA's cross-platform compatibility and convenient APIs for networking and multi-threading have won it a place in the business world. Java is the logically next step in the evolution of C++*

- **Program**
- **Software**
- **Compiler**
- **Interpreter**
- **Modular programming/Structured Programming**
- **Non structured programming**
- **Need of Programming language**
- **Difference between structured and object oriented programming**
- **Advantages of object oriented programming**
- **Object orientation**
- **Pillars of object oriented programming**
- **Classes**
- **Inheritance**
- **Polymorphism**
- **Encapsulation**

CLASSES

- One of the fundamental ways in which we handle complexity is abstraction.
- An abstraction denotes essential **properties** and **behaviors** of an object that differentiate it from other objects.
- The essence of OOP is modeling abstractions, using classes and objects.
- The hard part in this endeavor is finding the right abstractions.
- A class denotes a category of objects, and acts as a blueprint for creating such objects.
- A class models an abstraction by defining **properties** and **behaviors** for objects representing the abstraction.
- An object exhibits the **properties** and **behaviors** defined by its class.
- The properties of an object of a class are also called **attributes**, and are defined by **fields** in Java.
- A field in a class definition is a variable which can store a value that represents a particular **property**.
- The behaviors of an object of a class are also known as **operations**, and are defined using **methods** in Java.
- Fields and methods in a class definition are collectively called **members**.
- An important distinction is made between the **contract** and the **implementation** that a class provides for its objects.
- The contract defines what **services**, and implementation defines **how** these services are provided by class.
- Clients (i.e., other objects) only need to know the contract of an object, and not its implementation, in order to avail themselves of the object's services.

Compile and Execute Java Program

```
public class Mainclass
{
public static void main(String[] args)
{
   System.out.println(“Hello Quest”);
}
}
```

Compile

```
E:\jdk\bin>javac Mainclass.java
```

Execute

```
E:\jdk\bin>java Mainclass
```

Output

```
Hello Quest
```

Declaring Members: Fields and Methods

- Here is the class **Check**
- A class definition consists of a series of member declarations. In the case of the class **Check**, it has one field:
- **amount**, which is an integer to hold value of Check.

- The class Check has two methods that implement the essential operations on a Check:
- **setAmount**(int value) : To adjust the value of check.
- **getAmount**(): It retrieves and return the amount which is adjusted by setAmount() method.

  - The class definition also has a method-like declaration with the same name as the class.
  - Such declarations are called constructors. As we shall see, a constructor is executed when an object is created from class.
  - However, the implementation details in the example are not important for the present discussion.

```
class Check
{
private int amount=0;
     public int getAmount()
{     return amount; }
     public void setAmount(int amt)
{     amount=amt;}
}
public class Mainclass
{
     public static void main(String[] args)
{
        int amt=0;
        Check obj= new Check();
        obj.setAmount(200);
        amt=obj.getAmount();
        System.out.println("Your current amount is :"+amt);
        }
    }
```

Mobile class Example

- Write a program in java for class **Mobile** with **setBalance**(), **getBalance**() methods and **balance** as field.

```
class Mobile
{
private int balance=0;
public void setBalance(int blc)
{balance=blc;}
public int getBalance()
{return balance;}
}// end of Mobile class
class MainMobileclass
{public static void main(String arg[])
{
int blc=0;
Mobile obj=new Mobile();
obj.setBalance(1000);
blc=obj.getBalance();
System.out.println("The Balance of mobile is :"+blc);
}// end of main}// end of class
```

Passing String arguments in main

- Write a program in java for taking three arguments and display.

```
class StringArg
{
public static void main(String abc[])
{
System.out.println(abc[1]);
System.out.println(abc[2]);
System.out.println(abc[0]);
}
}
```

```
G:\jdk1.8\bin>javac StringArg.java
G:\jdk1.8\bin>java StringArg Anees Ahmed Soomro
```

Output

```
G:\jdk1.8\bin>
Ahmed
Soomro
Anees
```

Reference Materials

[1] H. M. Deitel, P.J. Deitel, “Java How To Program”, Prentice Hall.

[2] Ivor Horton, “Beginning Java 2”, Wrox Corp.

[3] Patrick Naughton, Herbert Schildt, “Java 2 : The Complete Reference”.

[4] Marty Hall, “Core Servlets and Java Server Pages”, Sun Microsystems Press/Prentice Hall.