CSE-2102
Object Oriented Programming

Dr. Muhammad Ibrahim
Assistant Professor
Dept. of Computer Science and Engineering
University of Dhaka

Lecture 3

Let’s start Java!

Topics

- Java’s lineage
- Platform independence
- Main features of Java
- Our first Java program

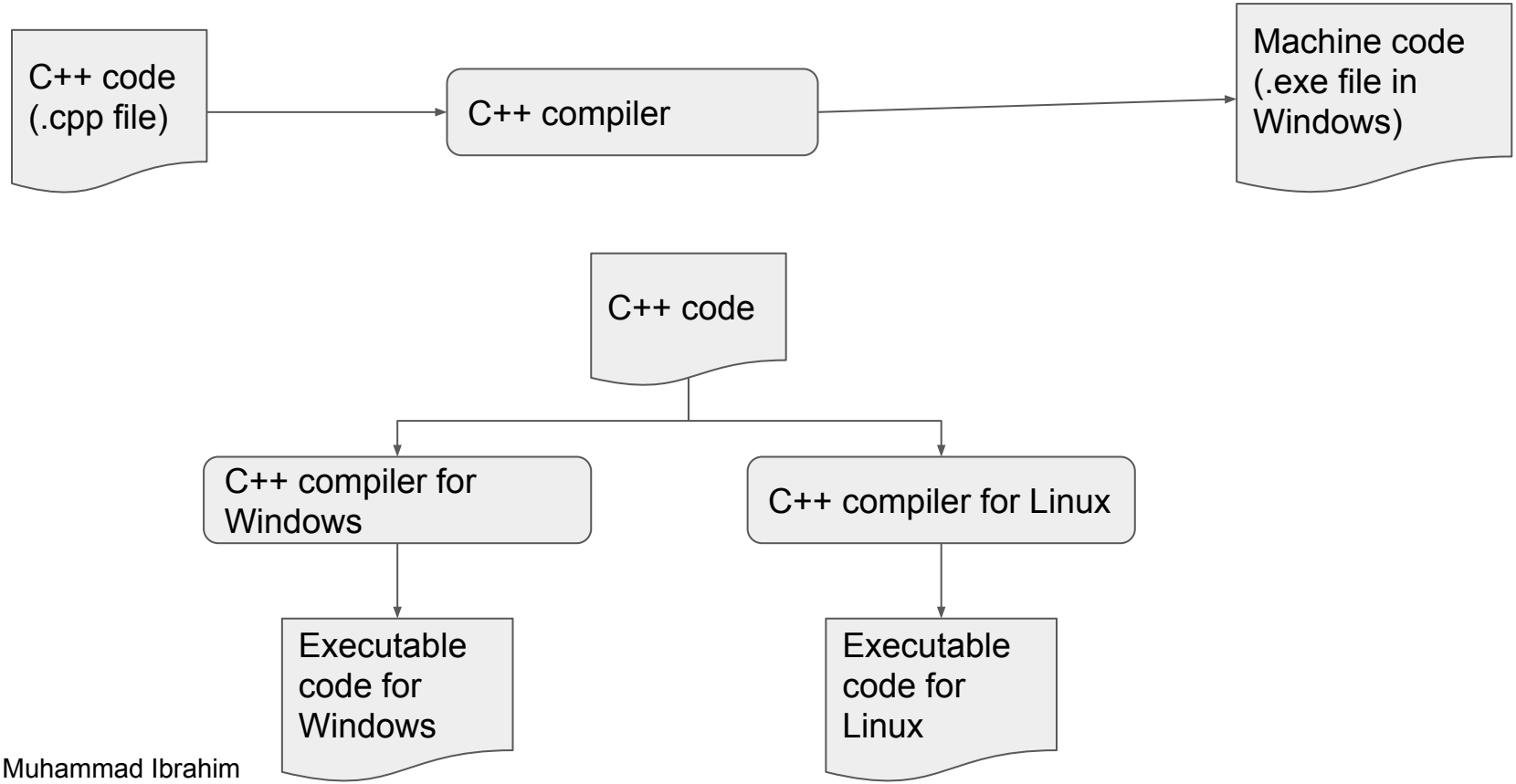
Java’s Lineage

- Java is related to C++, which is a direct descendent of C.
- Birth of C (early 1970s)
  - Need for a structured, efficient and high-level language.
- Birth of C++ (early 1980s)
  - Why need another language?
    - Increasing complexity of programs.
  - Object-oriented Programming (OOP) paradigm.
    - Although the first OOP was SIMULA designed in 1967, but did not get much popularity.
- Birth (1992-1995) and spread of Java
  - Why need another language?
    - Birth: Platform-independence (next slides), originally for software embedded in electronic devices.
    - Spread: due to the proliferation of Internet.

Platform Independence

- Aka architecture-neutral language, portable language, cross-platform language, multi-platform language etc.
- What do we mean by “platform”?
  - Different operating systems (OS) (Windows, Linux etc.)
    - Handles executable codes (produced by compilers) differently. E.g. system calls and file access instructions may be different in different OSs.
    - If the language is platform-dependent, different compilers are needed for different OS.
      - What is the problem to have different compilers?
        - Compilers are difficult to develop.
  - Different processor-families (x86, 6800 etc.)
    - Different instruction sets.
    - If the language is platform-dependent, different compilers needed for different processor-families.

Execution of a Platform-Dependent Language



Java's Forte: Platform-Independence

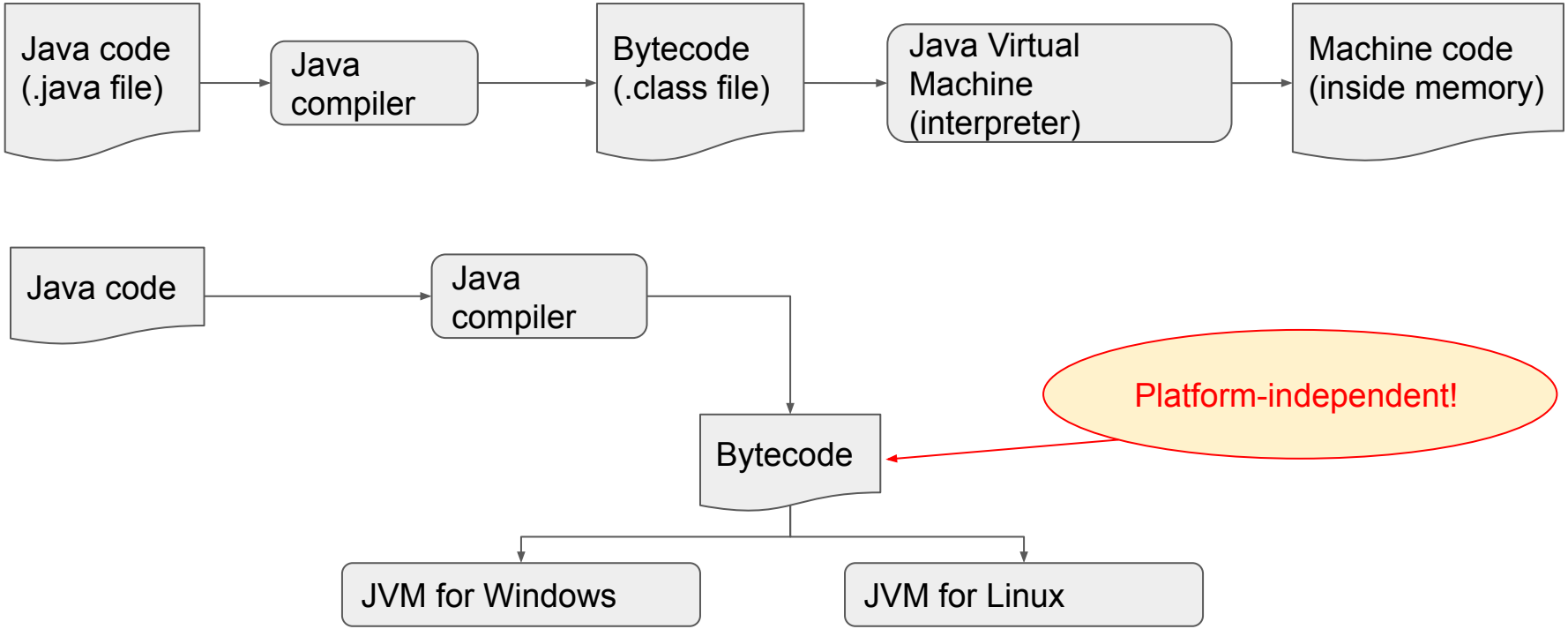
Bytecode

- A highly optimized set of instructions to be executed by the Java runtime system, known as Java virtual machine (JVM).
- Not executable code, but rather an intermediary code between source code and machine code.
  - Java assumes that everyone in the world is using the same machine that understands the bytecode.
- Does not depend on a particular platform.

Java Virtual Machine (JVM)

- Contains an interpreter for bytecode (among other components).
- Needs to be implemented for each platform.
- Easier to implement than a full-fledged compiler.

Java's Forte: Platform-Independence



Java and Internet

- Application programs Vs. Java applets (now obsolete)
  - An application is assumed to be executed under a specific operating system (OS), usually in which it is written.
  - Applet (now, however, obsolete in Java) is a special program meant to be transmitted over Internet and run by a Java-compatible web browser.
    - Operating systems do not necessarily need to be the same.
- Portability
  - Different OS, processors.
  - Java programs are portable.
- Security
  - Downloading programs from Internet is generally risky.
    - They might access your machine's resources.
  - However, Java's programs (applets) confine themselves in a specialized environment (i.e., JVM), without accessing other parts of the host computer.

Major Features of Java

- Simple
  - Many clumsy features of C/C++ are not present.
    - Eg. dynamic memory allocation, pointer arithmetic.
- Object-oriented
  - A natural approach to programming.
- Platform independent and portable
  - Bytecode, JVM.
- Robust
  - Many possible runtime errors are simply not allowed.

Major Features of Java (Contd.)

- Secure
  - Programs are confined inside the working environment of JVM, so can’t access other parts of the host machine.
- Relatively high performance
  - Although interpreted, the bytecode is in a highly optimized form.
  - Also, Just-in-Time (JIT) compiler does some extra optimization during runtime.

End of Lecture 3

Reading material: Chapter 1 of the textbook.

OOP Concepts

- Abstraction at object level
- Encapsulation
- Polymorphism
- Inheritance

Abstraction in OOP

- The core philosophy of OOP is to visualize the problem in terms of abstraction.
- While conventional (procedural) programming also allows this concept (using structure in C, for example), OOP provides a rich set of features to ease the implementation of this concept.

Three Concepts of OOP

- Encapsulation
  - The mechanism that binds data and code together.
  - It keeps data safe from outside interference.
- Inheritance
  - This mechanism allows for reusing codes and data.
- Polymorphism
  - This mechanism allows the programmers to use the same name for multiple (but related) entities.

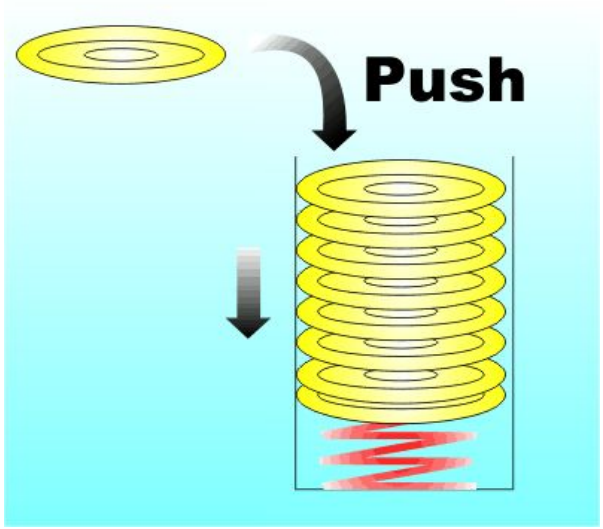
Java

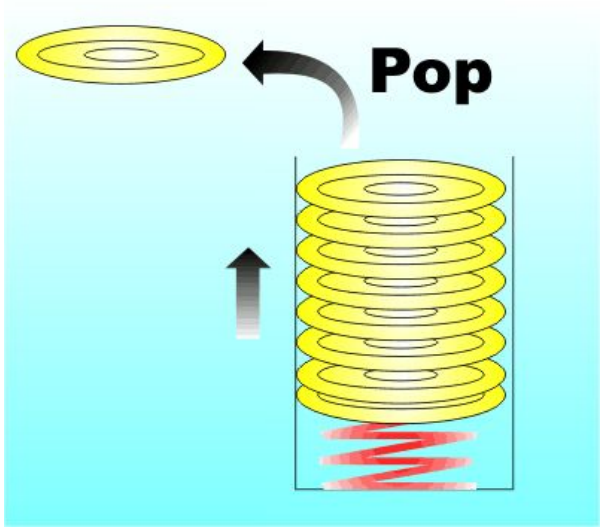
- Java is object-oriented.
- `class` is used to create object.
- Members
  - Can be data and code (or both)
  - Private: accessible only inside the class where it is declared.
  - Public: accessible by all classes of the program.
  - Any code of initialization can be put in the data section of a class

Real-Life Example

Image taken from Internet

Real-Life Example: Stack





Example: Stack

In C, we could write the following

```
void push (int st[], int tos, int num){
    st[tos] = num;
}
int pop(int st[], int tos){
    return st[tos];
}
```

```
main(){
   int stack[5];
   int tos=0;

   push(stack, tos, 45);
   tos++;

   push(stack, tos, 4);
   tos++;

   tos--;
   int item = pop(stack, tos);
   printf("%d", item);

   tos--;
   int item = pop(stack, tos);
   printf("%d", item);
}
```

Example: Stack

In C++, we write as follows

```
#include <iostream>
using namespace std;

class mystack {
  int stack[5];
  int tos = 0;
  public:
    void push(int num);
    int pop();

};
void mystack::push(int num){
        stack[tos] = num;
        tos++;
}
```

```
int mystack::pop(){
    tos--;
    return stack[tos];
}

int main(){
    mystack s1;
    s1.push(45);
    s1.push(31);
    cout << s1.pop() << endl;
    cout << s1.pop() << endl;
    return 0;
}
```

Example: Stack

In Java, we write:

```
public class myStack {

    private int stack[] = new int[5];
    private int tos = 0;
    public void push(int num){
        stack[tos++] = num;
    }
    public int pop(){
        return stack[--tos];
    }
    public void show (){
        System.out.println("Showing the
stack...");
        for (int i = 0; i < tos; i++){
            System.out.println(stack[i]);
        }
    }

}
```

```
class myStackDemo {
    public static void main(String[] args)
{
        myStack ob = new myStack();
        ob.push(4);
        ob.show();
        ob.push(7);
        ob.show();
        System.out.println("Popped: " +
ob.pop());
        ob.show();
    }

}
```

Encapsulation

- Mechanism that binds together code and data it manipulates
  - Keeps both safe from outside interference and misuse.
  - Thereby creating an *object* consisting of data and actions.
- Real life example
  - A book may have the following data: owner, number of pages read by someone.
  - Functions:
    - A book is bought from shop - at that time the owner is changed (so the data must be kept *private* so that the owner is changed only if the action of buying takes place).
    - A book is read - only at that time the number of pages read should be changed.
- A question on why do we need encapsulation anyway
  - “I’m the programmer, so how come other parts of my code **illegally** access a particular object’s data or code?”
    - Because when a program becomes very large, the programmer himself/herself can’t keep track of the codes.
    - Once a software is written, other people may get involved later in the project.

Encapsulation Contd.

- Within an object, data or code can be private or public.
  - Private elements can only be accessible by the elements of the object.
  - Public elements are open to other objects for being used.
    - Public elements typically provide controlled interfaces to private elements.
- Thus, an object is a variable of user-defined type.
  - Strange that a variable has both data and code!
  - Each time you define a new object, you're essentially creating a new (temporary) data type.
- C does provide some limited form of encapsulation.
  - By using library functions like printf, fopen.
    - fopen() creates and manipulates some local variables, but the programmer is unaware of these.
  - Java provides much more features on encapsulation.

Your First Java Program

Agenda for the first lab:

- Getting familiar with an IDE.
- Create, compile and run a Java program.
- Examine the bytecode.
- Smooth transition from procedural programming to OOP: solving some trivial problems

IDE to be used for the course (any one of your choice):

1) NetBeans
2) Eclipse
3) IntelliJ

And a lot of others!

- Instruction for installing an IDE is given in respective website.

End of Lecture 4.

End of lectures 1, 2, 3 and 4.