CSE-2102
Object Oriented Programming

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

Again, What is OOP Anyway?

So, by now it should be somewhat clear to you that OOP is a philosophy, a mental framework, a mindset, a practice, a culture, … … …

It is NOT a new technology. But it does spawns new technology (like new programming language).

This, once again, emphasizes the human aspects of engineering and technology: theoretically only machine code can solve almost all problems (except those “unsolvable” ones), but when it comes to reality (i.e. humans), they are too weak! They have too limited memory! So things need to be made easier for them.

Another Real-World Example

Consider a student of a university and a course.

They are two entities: former is a physical and the latter is abstract.

Each function MUST be associated with some entity - be it physical or abstract. This is rational because a program, no matter how complex, is nothing but manipulating some data in computer memory (in von neumann architecture).

Digression: There are other architectures besides von neumann architecture. See: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electronics/Implementing_a_One_Address_CPU_in_Logisim_(Kann)/01%3A_Introduction/1.03%3A_Von_Neumann_and_Harvard_Architectures

Topics

- Basics of OOP
- Basics of Java: comments, branching, iteration, block of code, variable scope, literals
- Data and expression types
- Pitfalls of assignment statement
  - Type conversion
  - Type casting

Abstraction in OOP

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

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.

Very Basics of Java

- Comments
- Basics of `main` function
- Console I/O
- Branching statement
- Loops
- Block of code
- Some lexical matters

Comments

Mostly same as C/C++

```
//a one line comment

/* multiple

line

comment.

*/
```

Basics of `main` Function

```
public static void main
```

- All Java application programs start from main function.
  - Like C/C++.
  - Applets (now obsolete) and some other schemes, however, don't have main.
- "`public`" is called the access specifier.
  - Has the same meaning as C++.
  - Public because main function should be called outside the class in which it is defined.
- It is a static method because it should be executed before creation of any object of the class.
  - Inside a static method, a non-static method (of the same class) cannot be called without using an object.
  - Static methods can be called without using objects.

Basics of `main` Function (Contd.)

- An important difference between main function of Java and that of C/C++:
  - In C/C++, main is a standalone function in the sense that it has no relationship with a class or anything.
  - In Java, however, main is always inside a class.

Console Input and Output

Point to note: Console inputs and outputs are not very common in real-life Java programs as these programs involve graphical user interfaces (GUI).

```
System.out.println("My first Java program.");
```

- Newline appended at the end of the string.
- `System` is a predefined class (details later).
  - It is by-default included in a program.
- `out` is a stream (object) connected to the output stream (details later).

Power of `System.out.println` function:

```
int i=10;
System.out.println("Value of i: " + i);
```

Console Input and Output

```
char ch;
c = (char)System.in.read();
```

- Reads characters, but returns as integers.
- Line buffered, so ENTER must be pressed.
- Also, this function may throw IOException (will be discussed in the middle of the course).

Another way to take console input:

```
Scanner sc = new Scanner(System.in);
 int num = sc.nextInt();
```

Many other ways are there !

Conditional Statements

- `if-else`
- `switch`

Iteration

- `for`
- `while`
- `do-while`

Block of Code

Same as C/C++

Lexical Matters

- Whitespace, identifier, literal, comment
  - All have mostly the same rules as C++.

So we have already covered the very basics of Java!

Data Types

Java is a strongly typed language. Part of Java's safety and robustness emerge from this property.

- Every variable and expression has a strictly defined type.
- In an assignment operation, no type-conversion or type-coercion of conflicting types takes place automatically.
  - For allowable types, however, automatic conversions do take place.
    - E.g.: assignment of an integer to a long is allowed.
    - But a double value cannot be assigned to a byte type - we need casting before this assignment.

Data Types

Primitive types/Simple Types

1. Integers: byte (8 bit), short (16 bit), int (32 bit), long (64 bit).

2. Floating-point: float (32 bit), double (64 bit).

3. Character: char (16 bit).

4. Boolean: boolean.

Latest addition in Java: “untyped” variable, type is assigned during initialization.

```
var i = 10; var j = 3.4;
```

Data Types

A few points to note:

- Java doesn't have `unsigned` type.
- Primitive types are not object-oriented features.
  - Java provides object-oriented features such as "Integer" (which will be discussed later).
- So why not make all primitive types object-oriented?
  - Because it will reduce the efficiency as objects take more CPU cycles than primitive types.
- As opposed to C/C++ where a string is an array of characters, in Java a string is an object, and is declared with the keyword `String`.
  - Using objects enables us to use a rich set of built-in functions on strings (will be discussed later in details).
  - However, using explicit character array is still allowed.

Data Types

- In C/C++ the size of a particular type depends on the platform. In Java, however, the size is fixed (32 bit for an integer) irrespective of the platform.
- In C/C++, a char is 8-bit long that contains the ASCII value of a character. Java, on the other hand, uses Unicode representation of characters - so a char in Java has 16 bits - this is to accommodate all different languages' symbols.
    - Note: the ASCII values of English letters are still the same in Unicode, so 'A' still has 65 as Unicode, for example. Unicode is a bit inefficient for the languages where it were not required e.g. English, but for the sake of global portability this price is paid.

Variables

- Declaration
- Initialization
- Assignment

Variables

- Inside block of code: same as C/C++.
  - A difference between C/C++ and Java: the following code segment is valid in C/C++ but not in Java:

```
{
  int i=1;
  {
    int i=2;
  }
}
```

- Inside a function: same as C/C++.
  - Global variable: allowed if it is inside a class.
- Inside a class: like C/C++
  - Accessible inside a class. Regarding access mode outside the class, this depends on the access mode of the variable (private or public etc.).
  - More on this will be discussed later.

A Note on Code Outside Methods (but inside class)

Allowed: Initializations, expression

Not allowed: loops, branching statements, method calls.

Question: When are such statements called?

Ans.:

A Note on Code Outside Methods (but inside class)

Allowed: Initializations, expression, block of codes.

Not allowed: loops, branching statements, method calls.
However, these are allowed inside the (static) block of code.

Question: When are such statements called?

Ans.:

1) For non-static fields, at the time of object creation of the class, before the constructor call.
2) For static fields, once for the class definition (not for each object creation). Just when main function begins execution (before any statement).

Example codes will be shown after studying constructor function.

Type Conversion and Casting

As stated earlier, in an assignment operation, no type-conversion or type-coercion of conflicting types takes place automatically.

For allowable types, however, automatic conversions do take place. E.g. assignment of an integer to a long is allowed. But a double value cannot be assigned to a byte type - we need to cast before this assignment.

Automatic Conversions

Two conditions in assignment operation:

1. Two types must be compatible. Numeric types (integer and floating point numbers) are compatible to each other, whereas numeric types are not compatible with char and boolean. Also, char and boolean are not compatible.

2. The destination size must be bigger than/equal to the source.

Coercion (Type Cast)

- In C/C++, type casting is done by default.
- In Java, however, it must be explicitly cast.
  - For example, the following code segment is valid in C/C++:

```
double d=12.3467;
int i = d;
```

  - But in Java explicit type cast is required as follows:

```
double d=12.3467;
int i = (int)d;
```

  - That's why Java is strongly typed language - you (i.e., the programmer) are totally aware of exactly what's being stored in an assignment operation.
  - From int to byte, the number that is stored is the modulo by the range of the byte.
  - For floating to integer, the fractional part of the number is truncated.

Java Primitive Type	Description	Java Data Range
boolean	unsigned 8 bits	0 (false) or 1 (true)
byte	signed 8 bits	-128 to 127
char	unsigned 16 bits	0 ('\u0000') to 65535 ('\uffff')
short	signed 16 bits	-32768 to 32767
int	signed 32 bits	-2147483648 to 2147483647
long	signed 64 bits	-9223372036854775808 to 9223372036854775807
float	32 bits	1.40239846e-45f to 3.40282347e+38f
double	64 bits	4.94065645841246544e-324 to 1.79769313486231570e+308

https://www.ibm.com/docs/en/i/7.2?topic=programs-cobol-java-data-types

Next Topics

Automatic type promotion
Basics of array
Basics of string
Pointer
Operators
Branching statements
Iterations
Classes

Automatic Type Promotion

In addition to assignment operation, in an expression, type casting may occur.

```
byte a=100, b=100;
int i = a*b;
```

Here the term `a*b` exceeds the range of byte.

To overcome this problem, Java automatically converts bytes, short and char to int, and integers to long.

The above may cause unexpected error, however. Consider the following code that raises a compile time error:

```
byte b=10;
b=b+2;//trying to assign an integer to byte! Error!
```

Automatic Type Promotion

Type Promotion Rules

1. In the beginning, all byte, short and char are converted to int.
2. If any one operand is long, the whole expression is made long.
3. If any one operand is float, the whole expression is made float.
4. If any one operand is double, the whole expression is made double.

Array

- Array in Java works a bit differently than that of C/C++.
- To declare an array, we write:

```
int a[10];
```

- but to allocate space for the array, we need to write:

```
a = new int[10];
```

- In C++, `new` operator is used only for allocating dynamic memory (i.e., memory from heap space as opposed to stack memory).
  - In contrast, in Java all arrays are dynamically allocated.

Array (Contd.)

- The initial values are automatically made 0 or equivalent, in particular, 0 for numeric values, null for reference type variables (remember that Java has no explicit pointers), and false for boolean values.
- Array access rules (indexing) are the same as C/C++.
- Multidimensional arrays are just like that of C/C++.

String

In Java, unlike C/C++, a string is not a sequence of character literals. A string in Java is in fact an object, not a primitive data type.

We'll discuss strings in details later in this course, but for the time being, how to use a string literal:

```
String str = "A text";
System.out.println(str);
```

Unlike C/C++, a string is not terminated by a null character. Rather, the null character is just like any other character.

```
String s = new String("a\0b");
System.out.println("s: " + s + ", len:" + s.length());
```

Output: s: ab, len: 3

Finally, C-like character arrays are still allowed (but seldom used):

```
public char str[] = new char[80];
str[0] = 'a';
str[1] = 'b';
System.out.println("str: " + str);  // output: ab
```

Pointer

- The concept of pointers is fundamental in almost all programming languages.
- In C/C++, programmers can explicitly manipulate pointers.
- In Java, however, programmers cannot explicitly manipulate pointers, but rather Java compiler manages the pointers.
    - Why you ask? Because one of the main merits of Java is that a Java program does not access "non-permitted" resources of a host computer.
    - More specifically, we know that a pointer can access any address of the host computer. But a Java program is not meant to have unlimited access to host computers. If Java were to allow programmers to use pointers, a Java program could not be said to be non-interfering with the host computer’s resources.
    - Also, lack of direct pointer arithmetic relieves many programmers.

Operators

- Four groups of operators: arithmetic, bitwise, relational and logical.
- Besides these, there are some special operators.
- Regarding arithmetic operators, the operands can be any data type except boolean.
  - char is a subset of int type, so char has no issues to be operands.
- Regarding relational operators, unlike C/C++, statements like `if(a)` and `if (!a)` are not valid in Java if a is not a boolean
  - You need to write the full version, i.e., `if (a != 0)` and `if (a==0)`.
  - However, if a is boolean, then this is fine.
  - In C/C++, true is any nonzero value and false is zero. In Java, in contrast, true and false are non-numerical values which don't have any relationship with zero or nonzero.
- An important operator: `instanceof`
  - `if (ob instanceof myclass) { … … }`

More on Branching Statements

- In C/C++, the variable based on which branching is performed can be only a few (int, char).
- In Java, however, a rich set of types can be used as branching variable: byte, short, int, char, enumerations (to come later) and (from JDK 7) Strings.
- Recall that if-else is more powerful than switch in that switch can check only for equality, whereas if-else can check for greater/less relationships etc. as well.
    - However, switch statements are easy to manage when we need a long if-else ladder.
    - Also, from the perspective of a compiler, a switch statement is faster to execute than an equivalent if-else.
    - Update: from JDK 17 (late 2021), flexibility of switch statement has been greatly empowered.
        - Check yourself.

Iteration

- Already discussed
- Recall that the following code is valid both in C++ and Java, but not in C:

```
for (int a : array) {   //array is an array of integers

    …

}
```

Jump Statements

- Conditional and unconditional break, continue and return.
  - Same as C/C++.
  - Note: too many jump statements harm the readability of your code, so use them wisely.
  - Labeled break and labeled continue are also possible in Java, but seldom used.
- In addition to these jump statements, an additional way to break the normal execution of a program is to use Java's exception handling mechanism (which will be taught later).
  - C++ also has this mechanism in a slightly different form.
- Unlike C/C++, Java doesn't provide a `goto` statement as `goto` dismantles the structured programming norm by jumping to here and there, thereby creating the so-called "spaghetti code" problem.

Next Topics

- Class
- Methods
- Reference variable
- Garbage collection
- Arrays of objects
- Overloading

Class

- Recall that Java is object-oriented.
- `class` is used to define object.
- Members
  - Can be data or code (or both)
  - Private members: accessible only inside the class where it is declared.
  - Public members: accessible by all classes.
  - Protected and default: to be discussed later.

Method definition:

```
ret-type func-name(param-list)
{
  //body of the function
}
```

Instantiation of objects (allocating memory):

```
class-name object = new class-name();
```

Calling (public) members from outside:

```
object-name.member
```

Class

```
class class-name {
    private int i; //private variable
    public char ch; //public variable

    public void f(){
        // code here…
    }

    public static void main(String args[]){
        class-name ob = new class-name();
        ob.f();
        //ob.i; // Compile time error
    }//main
}//class
```

Class

- Like any other OOP language, a class is at the heart of Java.
- However, unlike C++ where you may or may not use a class, in Java it is mandatory to use at least one class in a program, and any code you write must be a part of some class.
  - That is, unlike C++, there is no concept of "non-member function" in Java.
- So in this sense Java is a “strictly” OOP language as opposed to C++.
- A class defines a new data type which can be used to create objects, i.e., to allocate physical memory.
- Every object of a class has its own memory space.

Class (Contd.)

Some differences from C++:

1. In C++, you can write:

```
myclass a; //assuming no constructor function is present
a.data = 10;//assuming data is declared public in myclass
```

But in Java, you must write:

```
myclass a = new myclass();
a.data = 10;//assuming data is declared public in myclass
```

2. There is no semicolon after class definition.

3. In C++, we write `public` to make the data/code public. In Java, this is not the case. (public or no access specifier - details later)

Class (Contd.)

Notes:

- When we compile a Java program (i.e., .java file), a .class file is created for each of the classes present in the file.
  - Note that if you create multiple objects of the same class, there will exist a single class file for that class.

```
myclass a = new myclass();
myclass b = new myclass();
```

In the above, two objects of type `myclass` are created, but there will be a single *myclass.class* file in the directory. Natural, because the object creation takes space in memory (in runtime), not in directory.

Class (Contd.)

- Declaration versus allocation of memory
  - If we write `myclass ob`; this is a declaration, more specifically, a reference type variable `ob` that can point to an object of type `myclass`.
  - Then when we write `ob = new myclass();` the physical memory is allocated. `ob` actually stores the starting byte address of the memory allocated for the object.
  - In C++, you can manipulate a pointer variable -- pointer to a class object. In Java, however, you cannot manipulate the reference variable (e.g., `ob++` in the above example).
- Whenever an object of a class is created, a constructor function is called. (Discussion to come after a few slides.)
  - If the programmer doesn't write a constructor, a default constructor is called (whose job is to initialize the data with default values).
- Why primitive types such as `int`, `char` etc. are not implemented as objects?
  - The reason is efficiency - an object creation process incurs some overhead, i.e., spends some CPU cycle.

Access Control

- Commonly three access modes are used: public, protected and private.
- However, there is another one: using no modifier means yet another option that the member is accessible to the members of this package but not outside this package. (A package is a collection of classes.)
- protected is related to inheritance which will be discussed later.
- public and private modes work like that of C++.

Takeaways from the Discussion on Class

- Class is a construct that has members, i.e., data or code or both.
- It’s members can be private, public, default (no modifier), or protected
  - If public, they can be accessed outside their respective class in conjunction with an object
  - If private, they cannot be accessed outside their respective class
    - Private members can be indirectly accessed through public interfaces, i.e., functions.
- Class is a definition of a user-defined data type, NOT a memory allocation mechanism
  - Object declaration allocates physical memory.

Assignment of Object Reference Variables

```
myclass a = new myclass();
myclass b = a;
```

The above code snippet doesn't create a new object for `b=a` statement, but rather simply copies the address of object a to b so that both a and b **point to** (in Java jargon, **refer to**) the same physical object.

Now making `b=null` doesn't destroy the physical object, but rather a still refers to the physical object.

Output?

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        HelloWorld a = new HelloWorld();
        HelloWorld b = a;
        System.out.println(a.hashCode() + " " + b.hashCode() + "s: "
+ a.equals(b));
    }
}
```

So Far We've Learnt…

- When Java was invented and why?
- Very basic constructs of Java
  - Data types and assignment
  - Branch statement
  - Iteration
  - Methods
  - Class
    - How to define
    - How to access its members
    - private/public members
    - Memory allocation
- Benefit of the encapsulation mechanism of OOP over non-OOP
  - The stack example: how a real-life scenario can be translated into a program

End of Lectures 5 and 6.