CSE-2102
Object Oriented Programming

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

Topics

The keyword “static”

The keyword “final”

More on strings

Command-line arguments

Nested class

Variable length arguments

The Keyword “static”

- In Java every method must be inside some class.
  - So an object must be used to call a method outside the class where it is defined.
- However, sometimes we want to call a method without creating an object of the class to which the function belongs.
- In these cases, we write the keyword `static` in front of the function.
- Both the functions and variables can be declared static.
  - If we want to use a static method in a class where it is declared, we just use it by calling its name.
  - If we want to use it outside its class, we need to write classname.methodname format - but without creating an object.

The Keyword “static”

- The variables that are declared as static are actually global variables, i.e., all objects of this class share a single memory space.
- Restrictions that apply on static methods are as follows:
  - can call only static methods.
    - Non-static methods must be called using an object.
  - can only access static data.
    - Non-static data must be accessed using an object.
  - cannot use “this” or “super”

The Keyword “static”

- There is a mechanism to initialize static variables which is using a static block of code.
- This block is executed just after all the static statements (i.e., that involve static variables) are executed.
- The example of the following slide demonstrates this scenario.

The Keyword “static”

```
// Demonstrate static variables, methods, and blocks.
class static_usage {
     static int a = 10;
     static int b;
     static void func(int x) {
          System.out.println("x = " + x);
          System.out.println("a = " + a);
          System.out.println("b = " + b);
     }//func ends
     static {
          System.out.println("Initializing static block");
          b = a + 2;
     }//block ends
     public static void main(String args[]) {
          func(100);
     }
}//class ends
```

The Keyword “static”

```
class HelloWorld {
    static int a;
    static int b;
    static {
        System.out.println("a: " + a + ", b: " + b);
    }
    {
        b++;
        System.out.println("b in block: " + b);
    }
    public HelloWorld(){
        System.out.println("constructor");
    }
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        var ob = new HelloWorld();
        var ob2 = new HelloWorld();
    }
}
```

Output?

The Keyword “static”

```
class HelloWorld {
    static int a;
    static int b;
    static {
        System.out.println("a: " + a + ", b: " + b);
    }
    {
        b++;
        System.out.println("b in block: " + b);
    }
    public HelloWorld(){
        System.out.println("constructor");
    }
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        var ob = new HelloWorld();
        var ob2 = new HelloWorld();
    }
}
```

Output:

a: 0, b: 0
Hello, World!
b in block: 1
constructor
b in block: 2
constructor
==========

Lessons learnt

First, static codes (once per class)
Then, non-method codes (once per object, so object creation required)
Then, constructor (once per object, so object creation required)

The Keyword “final”

- A variable can be declared as final, and this means that the value of the variable is unchangeable.
- We must initialize a final variable in the declaration statement (or in the non-method block), or we must initialize it in a constructor of a class.
  - The point here is, final data must be initialized before/at the time of object creation.
- Methods can also be declared as final, but this meaning is drastically different which is related to inheritance (to be discussed later).

More on Strings

- Every string in Java is an object of class String.
  - Even string constant in `System.out.println("A text");` is also an object.
- A string in Java is immutable, i.e., not changeable.
  - Two related classes called StringBuffer and StringBuilder, however, allow strings to be changed.
    - When to use which? These libraries mostly differ in time and memory requirements.

A few operations:

- String creation: `String str = "A string";`
- Concatenation: `str = str1 + " more text" + str2;`
- `str.equals(str2), str.length(), str.charAt(index ).`
- Arrays of strings: `String str_arr[] = "ab", "cd", "ef";`

More on Strings

Output?

```
class HelloWorld {
    int a;
    public HelloWorld(int n){
        a = n;
    }
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        var ob1 = new HelloWorld(2);
        var ob2 = new HelloWorld(2);
        System.out.println(ob1.hashCode() + ", " + ob2.hashCode());
         String s1 = "S1a";
        String s2 = "S1a";
        System.out.println(s1.hashCode() + ", " + s2.hashCode());
         System.out.println(s1 == s2);
        s2 = s2 + " ";
        System.out.println(s1.hashCode() + ", " + s2.hashCode());
         System.out.println(s1 == s2);
    }
}
```

More on Strings

```
class HelloWorld {
    int a;
    public HelloWorld(int n){
        a = n;
    }
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        var ob1 = new HelloWorld(2);
        var ob2 = new HelloWorld(2);
        System.out.println(ob1.hashCode() + ", " + ob2.hashCode());
         String s1 = "S1a";
        String s2 = "S1a";
        System.out.println(s1.hashCode() + ", " + s2.hashCode());
         System.out.println(s1 == s2);
        s2 = s2 + " ";
        System.out.println(s1.hashCode() + ", " + s2.hashCode());
         System.out.println(s1 == s2);
    }
}
```

Output:

```
Hello, World!
295530567, 2003749087
81379, 81379
true
81379, 2522781
false
```

Command Line Arguments

Just like C/C++.

Nested Class and Inner Class

- Declared inside a class.
- Can access even private members of the surrounding class.
- But the surrounding class don’t have direct access to the members of the nested class, i.e., the enclosing class must use an object of the nested class to access its members.
- It is considered to be a member of the surrounding class.
- Usually non-static, but static nested class is also allowed (which is seldom used).
    - Inner class is a non-static nested class.
- Inner class objects can only be created inside the surrounding class, not outside it.
- Inner class can be created inside a deep block of the surrounding class.
- Application of inner/nested classes: in some specific scenarios.

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

Non-static nested class is called inner class

An example code:

https://www.programiz.com/java-programming/nested-inner-class#:~:text=Static%20nested%20classes%20are%20not,OuterClass.

Why Use nested class?

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

- It is a way of logically grouping classes that are only used in one place: If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.
- It increases encapsulation: Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
- It can lead to more readable and maintainable code: Nesting small classes within top-level classes places the code closer to where it is used.

Varargs: Variable Length Arguments (Self-Study)

- `int ... v` indicates that v is an array of integers that can take arbitrary number of integers as parameters.
  - varargs must be the last argument.
  - There can be only one varargs per function.
  - `int doIt(int a, int b, double c, int ... vals)`
  - `int doIt(int a, int b, double c, int ... vals, boolean stopFlag) { // Error!`
- Methods that use varargs can be overloaded.
  - However, ambiguity may arise because of overloading varargs.
- Final word on varargs: if not absolutely necessary, some programmers avoid the use of varargs mainly because of the possible ambiguity.

Overloading

- In OOP, polymorphism is mainly achieved using function overloading.
- Overloading mechanism in Java works just like that of C++.
  - Overloading of normal methods
  - Overloading of constructors

Polymorphism

- Mechanism that allows one name to be used for two or more related but technically different purposes.
  - In OOP, one name to be used for general class of actions.
- Example
  - In C, we have functions abs() for int, labs() for long int, and fabs() for floating point.
  - In Java (and C++), however, a single name abs() can be used to implement all these functions.
    - This is called function overloading.
- Thus polymorphism can be described as *one interface, multiple methods*.
- Helps to reduce program complexity.
- Note: In C++, this can also be applied to operators in addition to functions.
  - This is called operator overloading.
  - Developers of Java thought that this feature is not useful, so did not include it.

Function Overloading: Motivation

- Write a program that has three functions:
  - A function that prints 20 asterisks
  - A function that prints 20 of the character given as parameter
  - A function that prints n number of character given as parameter
- The three functions are somewhat similar in action, but the programmer has to remember three different names
  - Is there a more handy way?
    - Yes, function overloading!
- Same name can be used for different functions as long as number and/or type of parameters differ.
  - void drawline() void drawline(char ch) void drawline(char ch, int n)
- From the type and/or number of parameters, the compiler knows which version of drawline function is to be invoked.

Function Overloading

- Function overloading thus facilitates compile-time polymorphism.
  - Compiler automatically calls the correct version of overloaded function.
- Return type alone is not sufficient to overload functions.
  - Why? Check out the code given below:

```
// This is incorrect and will not compile.
int f1(int a);
double f1(int a);
.
.
.
f1(10); // which function does the compiler call???
```

Topics

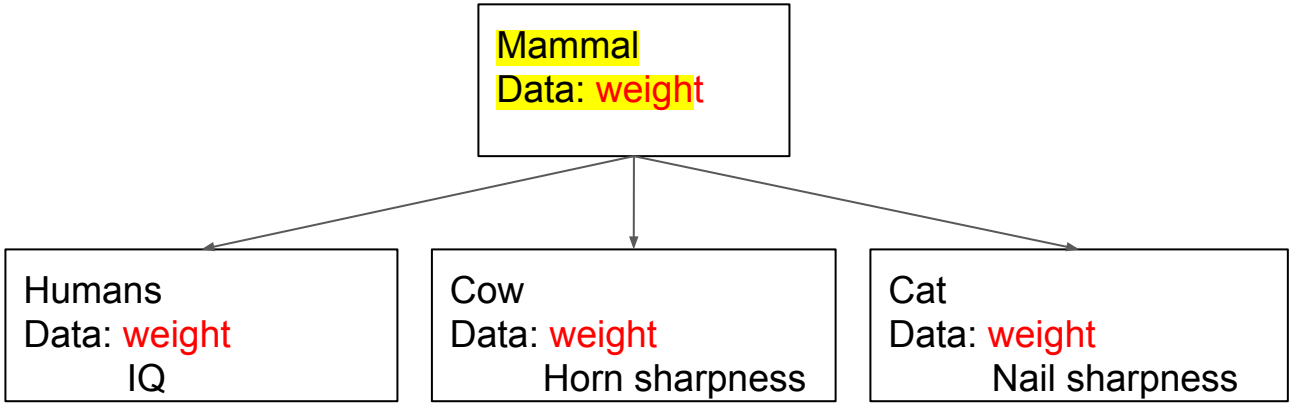
Inheritance
Access control with inheritance
Method overloading and method overriding
Referencing a subclass object with superclass variable
Using keyword “super”
Order of execution of constructors of superclass and subclass

Inheritance

- A mechanism of reusing code.
- Facilitates modular/hierarchical design of programs.
- Recall that everything in OOP is seen as objects, so objects may have commonalities.
    - If we view objects in a hierarchical fashion, then the top object in the hierarchy will contain common properties, whereas the objects of next layer will add specific properties to those common properties.
- Superclass and subclass (as opposed to the terminology used in C++ which were base class and derived class).
    - Note: There are some differences between Java and C++ in terms of inheritance - both in syntax and semantics.

Inheritance

Generalization
Specialization



- Super class: general properties.
- Sub class: adds specific properties to the general properties.

Member Access

C++ Vs. Java:
1) In C++, if no access mode is specified, it is private by default.
2) In C++, there is access mode of "extending" a class. So Java simplifies matters.

	Class	Subclass (package)	Package	Subclass (non-package)	Non-package
private	yes	no	no	no	no
default	yes	yes	yes	no	no
protected	yes	yes	yes	yes	no
public	yes	yes	yes	yes	yes

- Private members of base class still remain private to that class only.
  - Thereby maintaining encapsulation.
  - However, subclass may access them through public methods of superclass.
- Sometimes we may need some private members be accessible by the subclass and also classes of the same package, but by no other classes.
  - Protected members fulfill this purpose.
- Public members of superclass are inherited as public members of subclass.

Member Access

- A class cannot be private or protected, rather it must be default or public, but a nested class can be.
  - Recall that the members of a class can be: data, method, and nested class.
- Default class is not accessible outside the package.

```
//OPTIONAL: The following code can be used to see the access mode of a class variable.
import java.lang.reflect.*;
class myclass {
    int a=10;
    private int b = 100;
    protected int c = 1000;
    public int d = 1;
    public static void main(String[] args) {
        Field[] fs = myclass.class.getDeclaredFields();
        for (Field f : fs){
            int mod = f.getModifiers();
            if (Modifier.isPrivate(mod)){
                System.out.println(f.getName() + " is Private.");
            }
            // else if … …
        }
    }
}
```

End of Lecture 8