CSE-2102
Object Oriented Programming

Dr. Muhammad Ibrahim

Assistant Professor

Dept. of Computer Science and Engineering

University of Dhaka

Methods

- Unlike C/C++, functions in Java are usually called methods.
- Methods follow the same rules as C/C++ regarding return types and parameters.
- Recall that there is no provision for creating non-member functions in Java.

Constructor Methods

- In programming, initialization of variables is often needed.
- Almost each object needs some sort of initialization when created.
  - A bank account may need to set some initial fund
  - A library may need to set the number of available books
- Now, whenever we create an object, we could initialize its data, perhaps by calling a function (e.g., `init()`).
- However, a constructor function offers a handy way to achieve this.
  - A special member function that is automatically called by the compiler immediately after an object is created.
  - So we just need to put our code of initialization inside this function

Constructor Method

```
public class constructors {
    int num;
    String str;

    public constructors() {
        System.out.println("Inside constructor ... ");
        num = -1;
        str = "No string yet.";
    }

    public static void main (String args[]){
        constructors ob = new constructors();
    }
}
```

Constructor

- Constructor method takes the same name as the class.
- If the programmer does not write a constructor, Java compiler puts a default constructor.
- Constructors may take parameters but does not have a return type.
  - If you do put a return type, it will be treated as a normal method, not as a constructor.

```
public class constructors {
    int num;
    String str;

    public constructors(){
        System.out.println("Inside my constructor");
    }
    public int constructors(int n) {
        System.out.println("Inside method ... ");
        num = n;
        System.out.println("num inside method: " +
num);
        str = "No string yet.";
        return n;
    }
```

```
public static void main (String
args[]){
  constructors ob = new constructors();
  ob.constructors(5);
 }
}
```

Output: ?

Constructor

- Constructor method takes the same name as the class.
- If the programmer does not write a constructor, Java compiler puts a default constructor.
- Constructors may take parameters but does not have a return type.
    - If you do put a return type, it will be treated as a normal method, not as a constructor.

```
public class constructors {
    int num;
    String str;

    public constructors(){
        System.out.println("Inside my constructor");
    }
    public int constructors(int n) {
        System.out.println("Inside method ... ");
        num = n;
        System.out.println("num inside method: " +
num);
        str = "No string yet.";
        return n;
    }
```

```
public static void main (String
args[]){
  constructors ob = new constructors();
  ob.constructors(5);
 }
}
```

Output

```
Inside my constructor
Inside method ...
num inside constructor: 5
```

Constructors: Java Vs. C++

Parameterized constructors, return types etc. follow the same rules.

The default constructor of Java sets the default values to variables, i.e., 0 to numeric, null to character and false to boolean.

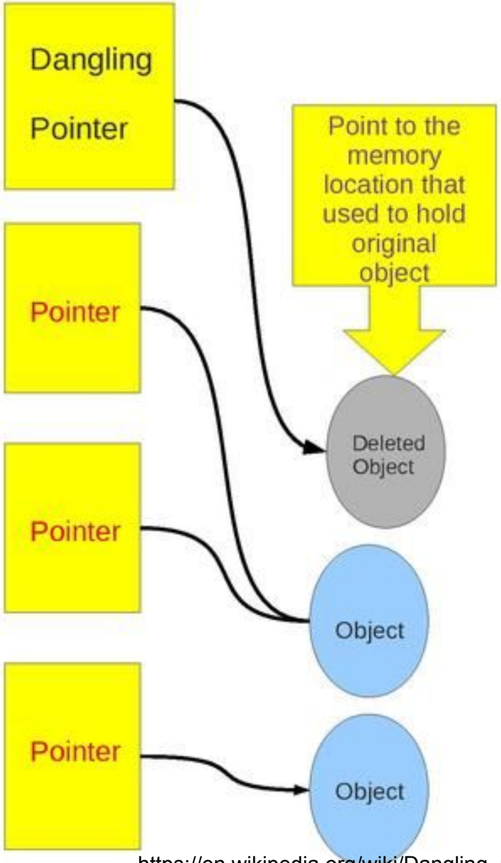
In C++, this is not the case, i.e., you might see garbage values if you print a variable without assigning an explicit value.

In addition, C++ has another similar function called destructor whereas Java does not.

“Destructor” Function?

- When it comes to an object going out of scope, is there any counterpart to constructor function?
- In C++, a destructor function can be defined that is invoked whenever an object is going out of scope, i.e., it can no longer be cited.
  - Benefits as well as side effects (mainly due to pointers).
- In Java, however, no such function is there.
  - Ques.: How then are Java program’s resources released?
  - Ans.: Using Java’s own garbage collection mechanism (to come after a few slides).

Dangling pointer in C/C++



https://en.wikipedia.org/wiki/Dangling_pointer

An Example Problem in C++

```
class myclass {
    public:
    int *p;
    int index;
    myclass(int i, int j) {
        index = i;
        p = new int;
        *p = j;
        cout << "Constructing object " << index << endl;
    }
    ~myclass() {
        cout << "Destructing object " << index << endl;
        delete p;
        cout << "delete performed."<< index << endl;
    }
};
int main()
{
    myclass x(1, 100), y(2, 99);
    x = y; //In C++, here a bitwise copy is made.

   return 0;
}
```

Output

```
Constructing object 1
Constructing object 2
Destructing object 2
Destructing object 2
Segmentation  fault
```

Passing Objects to Functions

- Just like other data types, objects can be passed to functions as parameters.
- Background (from your knowledge of C)
  - Call-by-value: e.g. passing built-in data types
  - Call-by-reference: eg. (1) passing arrays, (2) passing pointers.

Example

```
public class myclass {
    public void pass_args(yourclass ob){
        ob.num++;
        System.out.println("Hashcode inside method: " + ob.hashCode());
    }
    public yourclass return_ob(){
        yourclass ob = new yourclass();
        ob.num = 100;
        return ob;
    }
    public static void main(String args[]){
        myclass mob = new myclass();
        yourclass yob = new yourclass();
        System.out.println("Hashcode inside main method: " + yob.hashCode());

        System.out.println("num in yourclass: " + yob.num);
        mob.pass_args(yob);
        System.out.println("num in yourclass: " + yob.num);

        yourclass yob2 = mob.return_ob();
        System.out.println("returned: " + yob2.num);
    }
}
```

```
public class yourclass {
    public int num;
    yourclass(){
        num = 10;
    }
}
```

Reference Variable’s Benefit Over Pointer Variable

- When you use a reference parameter, the compiler automatically passes the address of the variable used as the argument; there is thus no need to type ‘&’ as required in C/C++.
- Furthermore, inside the function there is no need to use * to update the value since the compiler knows it.
- Important: the reference variable, unlike the pointer variable in C/C++, cannot be changed.
    - In the previous example, n++ inside the function would NOT mean the next location to n, but rather would mean the value of n is increased by 1.
- Reference is thus a more convenient and handy way than the pointer to implement *call-by-reference* concept.

Passing Objects to Functions: Java Vs. C++

- In Java, objects are passed using the *called-by-reference* scheme.
  - No bitwise copy of members, rather just a reference of the object is passed.
  - So changes in the object’s data inside the called function does change the object’s data of the calling function.
  - Since there is a single copy of the members of the object being passed as parameters, changes made inside the called function will be visible in the calling function.
- In C++, in contrast, objects are passed using the *called-by-value* scheme.
  - Bitwise copy of members.
  - So changes in the object’s data inside the called function doesn’t change the object’s data of the calling function.
  - To implement *call-by-reference* in C++, address of an object (in the form of explicit pointer) can be passed which requires the parameter to be a pointer to that object. A special reference type variable is also allowed.

Returning Objects from Function: Java Vs. C++

- In Java, when an object is returned by a function, a reference to the object is returned.
- In C++, in contrast, when an object is returned by a function, a temporary object is automatically created that holds the return value.
  - This (temporary) object is returned to the calling function.
  - After returning, this object is destroyed.
    - So possible side effects need to be considered if destructor function is defined.
  - Constructor function, however, is not called (like the case of call-by-value).
  - Important: many compilers, however, avoid creating the above-mentioned temporary object.

In a Nutshell: Using Objects as Parameters and Return Types

- A major difference between C++ and Java is as follows. In C++, when we pass an object to a function as a parameter, the *call-by-value* scheme is used, i.e., a bitwise (physical) copy of the object is created. In Java, in contrast, the *call-by-reference* scheme is applied in the sense that object variables are reference variable by-default (recall that copying a reference to another reference means that copying the address of the object), i.e., a reference to the object is created, so no new object is created.
  - Note, however, that in Java when we use primitive data types, it is passed using the call-by-value concept.
- Regarding returning an object from a method, the above discussion also applies here.

`this` Reference

- A special reference called “this” which is automatically passed to any method when it is called. `ob.f1();`
- It is a reference to the object that generates the method call.
- A common use of “this” is to remove ambiguity:

```
void myclass (int num){
    this.num = num;  //assume that num is a variable of myclass.
}
```

Output of the following code?

```
        boolean check (myclass ob){
            if (ob == this) return true;  return false;
        }
        System.out.println(mob.check(mob));//mob is a myclass type object
        myclass mob2 = new myclass(4), mob3 = new myclass(4);
        System.out.println(mob2.check(mob3));
```

Garbage Collection

- GC means reusing unused resources (memory).
- In C/C++, if we dynamically allocate memory (using `malloc()` or `new`), we need to explicitly free it using `free()` function or `delete` operator.
- In Java this is done automatically by the Java runtime system.
  - When no reference of an object exist, it is automatically deallocated.
    - So no way for the dangling pointer to be created!
  - Garbage collector runs periodically, without informing the programmers.
  - Yet, the programmer may explicitly call the garbage collector. This, however, is not too common in practice.

“finalize” Method

- In C++, we have the notion of a destructor function which is called automatically when an object is going out-of-scope.
- In Java although we don’t need to deallocate memory, we may need to perform some bookkeeping tasks such as closing a file.
- A “finalize” method is called just before the object is about to be deallocated by the garbage collector.
    - Since GC is called without the programmer’s notice, the programmer also doesn’t know when the finalize method is called. So if we want to release some resources at a specific time, we need to do it manually, i.e., writing a normal method - not the finalize method.
        - Here lies a difference between C++’s destructor and Java’s finalize - a destructor is called when an object goes out-of-scope, so we know at what point of code it is called (and so you can work out the output). But a finalize method is not necessarily called when an object goes out-of-scope, but rather the programmer doesn’t know when it will be called (by the garbage collector).
    - So is C++ more powerful than Java in this regard? Not necessarily, as we dive into details of Java we’ll see that the need of a destructor function is not felt at all in Java.

Arrays of Objects

- Since the objects are (user-defined) data types, like other variables you can create array of objects.
- Syntax is the same as others arrays.

```
myclass ob[] = new myclass[4];
ob[0] = new myclass();
ob[1] = new myclass();
myclass nob = new myclass();
myclass ob2[] = {new myclass(4), new myclass(3)};
//   myclass ob3[] = {4, 5}; // ERROR in Java, but OK in C++
System.out.println(nob.getClass());
System.out.println(ob.getClass() + " " + ob.hashCode());
System.out.println(ob[0].hashCode() + ", " + ob[1].hashCode());
System.out.println(ob[0].getClass());
```

Note: Java object’s physical address is not usually visible (however, advanced libraries may do it).
Hashcodes are just “symbolic identifiers” for objects used by JVM to allocate physical memory.

Digression: try these yourselves

- `==` operator //checks references (i.e., hashcodes) of the two objects
- `instanceof` operator
- `equals()` method //checks the contents of the two objects
- `compareTo()` method
- `compareToIgnoreCase()` method
- `compare()` method
- `hashCode()` method
- `getClass()` method
- `toString()` method //default implementation returns classname@hashcode_ in_Hex.

… with both your defined objects and string objects.

Array of Objects (contd.)

```
myclass twob[][] = {
                     {new myclass(3), new myclass(), new myclass()},
                     {new myclass(2)}
                    };
System.out.println(twob + ", # rows: " + twob.length);
System.out.println("# cols in 1st row: " + twob[0].length +
                      ", # cols in 2nd row: " + twob[1].length);
```

“new” Operator

- Alternative to malloc() function of C, in Java we have new operator.
- Safer (less chance of syntax error) and more convenient way for dynamic memory management.
- Advantages of new over malloc()
    - No sizeof is required.
    - No type-casting is required.
        - In C, no casting is required for malloc(), but in C++ it does.
    - Possible to initialize during memory allocation.
- Note: why dynamic memory (using new)?
    - Much more memory allocation than static allocation is possible
    - 10000000 sized integer array was not found to be possible in my experiment, but with dynamic memory it was found to be fine!

Recursion

Recursion in Java follows the same rules as in C/C++.

End of Lecture 7