CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Java Language Constructs II

Department of Computer Science
University of Maryland, College Park
Three Levels of Copying Objects

Assume y refers to object z

1. Reference copy
   - Makes copy of reference
   - \( x = y; \)

2. Shallow copy
   - Makes copy of object
   - \( x = y.clone(); \)

3. Deep copy
   - Makes copy of object z and all objects (directly or indirectly) referred to by z
Cloning

• Cloning
  • We can create a copy of an object using the clone() method
• The **Object** class provides a clone() method that provides shallow copying
  • See prototype at
• To clone objects of a particular class, override the clone() method and call the Object class **clone** method. If the fields of the class are primitives or references to immutable objects, there is nothing else you need to do; otherwise you may need to duplicate objects referred by reference instance variables
Cloning

• Regarding overriding `clone()` method (Object class)
  • When overriding a method in Java, you can define the return type to be a subtype of the return type of the method being overridden. This is known as a covariant return type.
  • The above means that when defining a clone() method for a class the return type of the overriding clone() method can be changed to the class type. For example, if you are defining the clone() method for a `Mouse` class, the method we are overriding is `protected Object clone()` and we will override it with `public Mouse clone()`
Cloning

• A class needs to implement the Cloneable interface if it calls the Object class clone() method. If a class calls the clone method, but it does not implement the interface, the exception CloneNotSupportedException will be generated.

• From the Java API
  • CloneNotSupportedException - Thrown to indicate that the clone method in class Object has been called to clone an object, but that the object's class does not implement the Cloneable interface.

• The Object class clone() method is defined as protected.

• Example: cloning package
  • Mouse.java, Computer.java, SuperComputer.java
Garbage Collection

• Concepts
  • All interactions with objects occur through reference variables
  • If no reference to object exists, object becomes garbage (useless, no longer affects program)

• Garbage collection
  • Reclaiming memory used by unreferenced objects
  • Periodically performed by Java
  • Not guaranteed to occur
  • Only needed if running low on memory
  • Suggesting JVM to do garbage collection using System.gc() method
    • https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/System.html#gc()
Destructor

- Description
  - Method with name `finalize()`
  - The name is misleading as it does not destroy the objects
    - Contains action performed when object is about to be freed
  - Returns void
  - Invoked automatically by garbage collector
    - Not invoked if garbage collection does not occur
  - Usually needed only for non-Java methods
  - `finalize` is only called if the object is garbage collected, therefore there is no guarantee that `finalize` will always be called as an object might not be garbage-collected
  - Destructors are used a lot in C++

- Example
  ```java
class Foo {
    void finalize() { … } // destructor for foo
}
```
Initialization Block

• **Definition**
  - Block of code used to initialize static & instance variables for class

• **Motivation**
  - Enable complex initializations for static variables
    - Control flow
    - Exceptions
  - Share code between multiple constructors for same class
Initialization Block Types

• Static initialization block
  • Code executed when class is loaded
  • A class is loaded when it is needed and it is loaded only once (therefore static blocks only executed once)

• Initialization block
  • Code executed when each object created
    (at beginning of call to constructor)

• Example

```java
class Foo {
    static {
        A = 1; // static initialization block
    }
    {
        A = 2; // initialization block
    }
}
```
Instance Variables Initialization

- Instance variables may be initialized
  - At time of declaration
  - In initialization block
  - In constructor

Order of initialization

1. Declaration, initialization block (in the same order as they appear in the source code)
2. Constructor – overrides any other initializations

**Example:** staticBlock package → VariableInitialization.java

**Example:** staticBlock package → Person.java, PersonDriver.java
  - By using a static block we only need to create a single MILLENIUM object

**Example:** nonStaticBlock package → Employee.java