
MAHARISHI UNIVERSITY of MANAGEMENT
Engaging the Managing Intelligence of Nature
Computer Science Department
CS401 Modern Programming Practices (MPP)
Najeeb Najeeb

Lesson 6

Creating & Destroying Objects

najeeb@mum.edu

Ex. 3436

Creation

UML to Code

- How to represent cardinality.
- How to represent associations.
- How to represent inheritance.
- How to convert code to UML.

Creating Objects

- What are my options
  - Constructor.
  - Static factory method.
- Why use a static factory method?
  - They may have meaningful names.
    - Eg. You wish to create a number (random between a range and another one where it is a prime number within a range).
  - They don't have to create an object every time.
  - They can return objects of a subtype.
  - Can use type inference
    - Map<String, List<String>> map = new HashMap<String, List<String>>()
    - Map<String, List<String>> map = HashMap.newInstance();
  - You need to maintain relations (associations).

Factory methods

- Disadvantages
  - Classes without public or protected constructors cannot be subclassed. (may be a good thing)
  - They are not easily distinguishable from other static methods.
    - So we come up with a convention
      - getInstance
      - newInstance
      - getType
      - newType
      - valueOf
      - of
- Consider your options, you may need a constructor or factory method.

Many optional parameters

- Constructors and factory methods do not scale.
- With more an more optional parameter in your class how can you provide instance construction options:

Telescoping

- Constructor chain
- How to build it
  - The constructor with required parameters.
  - Constructor with one optional parameter.
  - Build up till you reach all optional parameters.
- Problems?
  - Hard to write client code.
  - Hard to read client code.
  - Forced to pass values for parameters you don't use.

JavaBeans

- The JavaBean approach, using setters.
  - Easier to write and read.
- How to build it?
  - Create a constructor with required parameters.
  - Create setter methods for each optional property.
- Problems?
  - Construction is split across multiple calls.
  - The object may be in an inconsistent state partway.
  - Enforces classes to become mutable.

Builder

- Builder; provides the safety of telescoping, and the readability of JavaBeans.
- How to build it
  - Create a static Class inside your class.
  - Make the constructor take the mandatory parameters.
  - Create a method for each property, that returns the builder itself.
  - Create a factory method; that invoke constructor on the Original Class taking the builder as a parameter.
  - The Original Class constructor takes a builder and creates an instance based on the properties of the builder.
- A generic builder can be created

```
public interface Builder<T> {
        public T build();
}
```

Enforce noninstantiability

- Why would you do that?
  - Utility classes (only have static methods).
- How do you do that?
  - Abstract (but they can be extended, intended for extension)
- private constructors.

Avoid creating unnecessary objects

- Strings
  - String s = new String("Hello");
  - String s = "Hello";
- Constant initialization
  - Method.
  - Static initializer block.
- Autoboxing (watch out for unintentional autoboxing)
  - Long
  - long
- Note: creating objects (Classes) to enhance clarity, simplicity, or power of program is generally a good thing.

Eliminate obsolete object references

- When you are given garbage collection you may think you never need to think about memory management.
- Stack example.
  - Is there a “memory leak” ?
- Avoid unintentional object retentions.
- Nulling object references should be the exception rather than the norm.
  - Let objects fall out of scope.
  - Perform when class manages its own memory.
- Another common source of memory leak is caches.

Avoid finalizers

- Finalizers are unnecessary, unpredictable, and dangerous. So avoid most of the time.
- Finalizers are not like C++ destructors.
- Use try finally block to reclaim nonmemory resources.
- No guaranty of execution.
- Do not perform time critical operations in finalizers.
  - Never close files in a finalizer. (JVM allows only a number of files to be open).
- Exceptions thrown in finalize are not even caught.
- It is slower to create and destroy an object with finalize.

Finalizer alternate

- Instead of using finalize you should
  - Provide explicit termination methods.
    - Some internal state must be kept, and IllegalStateExcpetion thrown in unsafe usage.
  - Invoke termination methods inside try finally to ensure invocation.

When to use finalizers

- To invoke termination methods (better late than never) safety net approach.
  - But still weigh the cost very well.
  - Also make sure to log this, since it indicates an error in the client code.
  - Maybe a good idea is to do this during testing phase (alpha releases).
- Finalize native peer classes (native delegates).
- Always call super.finalize() in the end, since finalizers do no chain.

Homework

- What is the difference between StringBuffer & StirngBuilder.
- Write a program to show how string concatenation could cause unpredictable results.