CS601: Principles of Software Development

Nested Classes Continued.

Olga A. Karpenko

Parts of this presentation is based on the materials of Prof. Engle.

Announcements

- Lab 3 Part 1 due tonight
- Lab3 Part 2 will come out tonight
- Quiz on Wednesday

Survey Results: Comments

- More examples & coding exercises
- Students should work on design
  - no starter code
- Bigger Projects, Harder Labs
  - Extra Credit on Labs
- More information about edge cases

Survey Results: Comments

- Too much time spent reviewing the basics
- TA office hours on Tue or Th
- Talk about Code Style
- Team Projects
- More tools that are used in industry: git/ github, junit, continuous integration, etc.

Lab 3 Part 2

- Not allowed to use any classes from Java's built-in concurrent package
  - like Executors, ExecutorService etc
- Required to use provided WorkQueue
- Do not shutdown the threads after they are done. Not allowed to use awaitTermination().
- Keep track of the # of tasks. Print to file only the # of current tasks is 0
- Make your code efficient: each thread should add reviews to the **local** ThreadSafeHotelData
- Later merge with the "big" ThreadSafeHotelData

Static Nested vs Inner Classes

- Favor static nested classes over inner
  - Less coupling
  - Easier to use, easier to understand code
  - *Unless the nested class needs access to variables of the outer class*
- Typical use of a static nested class: a public helper class
- Typical use of an inner class: Iterator class

Anonymous Inner Classes

Anonymous Inner Class

- A nested class without a name
- Defined and instantiated at the same time
- Used for classes defined & used only once
  - Makes your code concise

Syntax

```
new SomeSuperClass(args) { body }

or

new SomeInterface() { body }
```

Example 1: Multithreading

```
Runnable r = new Runnable() {
      @Override
      public void run() {
            System.out.println("Hello!");
      }
}; // don't forget the semicolon!
```

http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html

Example 1: Multithreading

```
Runnable r = new Runnable() {
      @Override
      public void run() {
            System.out.println("Hello!");
      }
}; // don't forget the semicolon!
```

- Created an object of an anonymous class that
  - implements Runnable
  - overrides the run() method to print Hello.

http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html

Example 1: Multithreading

- Java 8 syntax

```
Runnable r = run() -> {
            // Code

};
```

- See `Deadlock.java`

http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html

Example 2

```
interface HelloWorld {
    public void greet();
}
```

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

Example 2

```
HelloWorld frenchGreeting = new HelloWorld(){
        String name = "tout le monde";

        public void greet() {
            System.out.println("Salut " + name);
        }
};
```

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

Example 3: Comparator

```
Comparator<Color> comp = new Comparator<Color>() {
 @Override
 public int compare(Color s1, Color s2) {
    return s1.getColor().compareTo(s2.getColor());
 }

}; // Assume that getColor() returns a string


Set<Color> colors = new TreeSet<Color>(comp);
colors.add(new Color("red"));
colors.add(new Color("green"));
colors.add(new Color("blue"));

System.out.println(colors);
```

Anonymous Classes

- Anonymous inner classes can not have explicit constructors
- Possible solutions:
  - Use final local variables
  - initializer method that takes parameters and returns the instance of the class
  - Extend a superclass that has a constructor

  - `AnonymousClassExample.java`
  - `ColorComparatorWithAnonymousClass.java`

Local Classes

Local Class

- Defined within a method of an outer class
- `LocalClassExample.java`, `HelloWorldAnonymousClasses.java`

References

- http://tutorials.jenkov.com/java/nested-classes.html