CISC 3115 TY3

C30b: Inner Class, Anonymous Class, and Lambda Expression

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Concept of Stage, Scene, and Scene Graph
  - How to build views, representing visual elements of a Graphical User interface
  - Handling events
    - Event
    - Event dispatcher chain
    - Registering handlers and handling events
- To discuss
  - Writing event handlers efficiently: inner class, anonymous inner class, and Lambda expression

Writing an EventHandler

- Always follow the pattern:

```
class MyEventHandler implements
EventHandler<MouseEvent> {

    public void handle(MouseEvent) {

            …

    }

}
```

Writing Event Handler: Inconvenience

- Using the pattern, we have to create many small classes, each has only a few lines of code
- Soon, we will have many classes and files
  - MyEventHandler.java
  - HerEventHandler.java
  - HisEventHandler.java
  - TheirEventHandler.java
  - OurEventHandler.java
  - YourEventHandler.java
  - DoNotKnowHowToNameEventHandler.java
  - TiredOfNamingItEventHandler.java
  - ThereIsEvenMoreEventHandler.java
  - ….

Using Nested Class

- Nested class
  - In Java, a class can be defined anywhere within a class
- Java has 4 types of nested classes

Nested Class

- Java permits one to define a class within another class. Below are 2 of 4 types:
  - Inner class (Non-static nested class)

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

  - Static nested class

```
class OuterClass {
    ...
    static class StaticNestedClass {   ...   }
}
```

Using Nested Class

- Logically grouping classes that are only used in one class
- Can increase encapsulation
- Can lead to more readable and maintainable code

```
class B {
    int c;
}
class A { // B only used in A
    B b = new B();
    b.c = 2;
}
```

```
class A {
    class B {
        int c;
    }
     B b = new B();
     b.c = 2;
}
```

Inner class

- An inner class is a member of the outer class
  - have access to other members of the enclosing class, even if they are declared private.
  - An inner class can be declared private, public, protected, or package private.
  - However, the outer classes can only be declared public or package private

Inner Class: Member of Outer Class

- An instance of the inner class is a part of an instance of the outer class
  - How about create an object of the inner class

Inner Class: Member of Outer Class: Examples

- Which one is correct?

```
class A {
    void method() {
        B b = new B();
    }
   class B { // B only used in A
    }
}
```

```
class A {
   void method() {
        B b = this.new B();
    }
   class B { // B only used in A
    }
}
```

```
class A {
    static void method() {
        B b = new B();
    }
   class B { // B only used in A
    }
}
```

```
class A {
    static void method() {
        A a = new A();
        B b = a.new B();
    }
   class B { // B only used in A  }
}
```

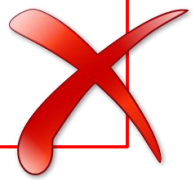
Inner Class: Member of Outer Class: Examples

- Which one is correct?

```
class A {
    void method() {
        B b = new B();
    }
   class B { // B only used in A
    }
}
```

```
class A {
   void method() {
        B b = this.new B();
    }
   class B { // B only used in A
    }
}
```

```
class A {
    static void method() {
        B b = new B();
    }
   class B { // B only used in A
    }
}
```



```
class A {
    static void method() {
        A a = new A();
        B b = a.new B();
    }
   class B { // B only used in A  }
}
```

Static Nested Class

- A static nested class is associated with its outer class
  - It belongs to the outer class, not to an object of the outer class.
  - Behaviorally a top-level class that has been nested in another top-level class for packaging convenience.

Static Nest Class: Examples

- Which one is correct or wrong?

```
class A {
    void method() {
        B b = new B();
    }
   static class B { // B only used in A }
}
```

```
class A {
    void method() {
        B b = new A.B();
    }
   static class B { // B only used in A }
}
```

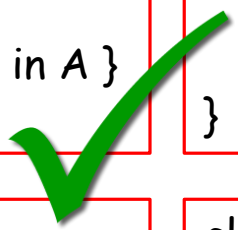
```
class A {
    static void method() {
        B b = new B();
    }
   static class B { // B only used in A }
}
```

```
class A {
    static void method() {
        B b = new A.B();
    }
   static class B { // B only used in A }
}
```

Static Nest Class: Examples

- Which one is correct or wrong?

```
class A {
    void method() {
        B b = new B();
    }
   static class B { // B only used in A }
}
```



```
class A {
    void method() {
        B b = new A.B();
    }
   static class B { // B only used in A }
}
```

```
class A {
    static void method() {
        B b = new B();
    }
   static class B { // B only used in A }
}
```

```
class A {
    static void method() {
        B b = new A.B();
    }
   static class B { // B only used in A }
}
```

Nested Class: More

- Java permits one to define a class within another class. Below are the other 2 of 4 types:
  - Anonymous class

```
class OuterClass {
    ...
    SomeParentClass c = new SomeParentClass() {   ...   };
}
```

  - Lambda Expression

```
class OuterClass {
    ...
    SomeFunctionalInterface inf = (p1, p2) -> {   …do sth with arguments p1 and p2   };
}
```

Local Class

- Classes defined within a block
  - A block: what between a pair of balanced braces i.e., { }
  - A block can be used anywhere a single statement is allowed.

```
class OuterClass {
    ...

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

Local Class: Characteristics

- Local classes are similar to inner classes
  - A local class has access to the members of its enclosing class.
- In addition, a local class has access to final or effectively final local variables
  - Final variables: e.g., final int a;
  - Effectively final, e.g., int a = 1; but variable "a" never changes after initialization
- It can access the method's parameters
- However,
  - cannot declare static initializers or member interfaces in a local class.
  - can only have static members only when they are constants (final static …)

Using Inner or Local Class: Handling Event

- Observe the example

Using Anonymous Class: Motivation

- If we only create one instance of an inner class, we still have to write

```
class MyInnerClass implements SomeInterface {
    …
}
```

- Using anonymous class to reduce these boiler plate code

Anonymous Class

- Essentially, a local class without a name
- Created by declaring and instantiating a class at the same time
- Use it when need a local class only once

```
class OuterClass {
   ...

   {  ...
      ParentClass a = new ParentClass() { ...}
   }
      ...
}
```

Anonymous Classes are Local Classes

- It has access to the members of its enclosing class.
- In addition, it has access to final or effectively final local variables
  - Final variables: e.g., final int a;
  - Effectively final, e.g., int a = 1; but variable "a" never changes after initialization
- It can access the method's parameters
- However,
  - cannot declare static initializers or member interfaces in a local class.
  - can only have static members only when they are constants (final static …)

Using Anonymous Event Handler: Example

- Observe the example

Functional Interface

- Any interface that contains only one abstract method
  - Since Java 8, a functional interface may contain one or more default methods or static methods (these are not abstract methods)

Use Functional Interface

- In your own program, sometimes a functional interface is a better design choice
- More often, you use functional interfaces because some Java API methods require them
  - Examples:
    - https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Functional Interface and Anonymous Class

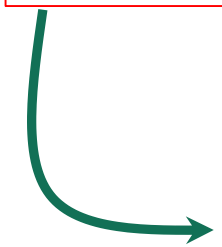
- You can declare and instantiate a local class, or more often an anonymous class
- Example

```
ArrayList<Person> personList = new ArrayList<Person>();
Arrays.sort(personList, new Comparator<Person> {
  @Override
   public int compare(Person lhs, Person rhs) {
     // buggy (what if rhs is null?)
     return lhs.getName().compareTo(rhs.getName());
   }
}
```

Lambda Expression

- A simple way to declare and instantiate a class

```
ArrayList<Person> personList = new ArrayList<Person>();
Arrays.sort(personList, new Comparator<Person> {
  @Override
   public int compare(Person lhs, Person rhs) {
     // buggy (what if rhs is null?)
     return lhs.getName().compareTo(rhs.getName());
   }
}
```



```
ArrayList<Person> personList = new ArrayList<Person>();
Arrays.sort(personList, (lhs, rhs) -> lhs.getName().compareTo(rhs.getName())}
```

EventHandler is a Functional Interface

- We can use Lambda expression to reduce boiler plate code further when writing event handlers

EventHandler in Lambda Expression: Example

- Observe the example

Questions?

- Nested class
  - Static inner class
  - Inner class
  - Local class
  - Anonymous class
  - Lambda expression