Victoria University
of Wellington, New Zealand
Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa

SWEN221:
Software Development

20: Nested Classes

Outline

- Why Nested Class
- Inner Classes
  - Static vs non-static
- Relationship between Enclosing Classes
- Local Classes
- Anonymous Classes

Why Nested Classes?

- We want a **iterable list** class `MyList<T>` with a special iterator
- The **special iterator** `EvenIter<T>` scans only the even indices
- The iterator is so special that **NO** other list classes is likely to use it

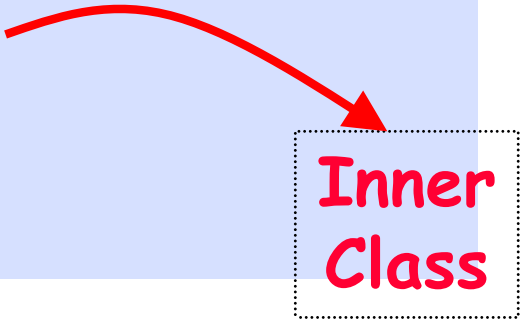
- Define `EvenIter<T>` as an **inner class** inside `MyList<T>`

Why Nested Classes

- Increase logic
- Increase encapsulation
- More readable and maintainable code

Inner Classes: Example

```
public class MyList<T> implements Iterable<T> {
  private T[] data;

  public MyList(T[] d) { data = d; }

  public Iterator<T> iterator() { return new EvenIter(); }

  private class EvenIter implements Iterator<T> {
    private int pos = 0;

    public boolean hasNext() { return pos < data.length; }

    public T next() {
      T next = data[pos]; pos += 2;
      return next;
    }
}}
```



Inner Classes: Example

```
public class MyList<T> implements Iterable<T> {
  private T[] data;

  public MyList(T[] d) { data = d; }

  public Iterator<T> iterator() { return new EvenIter(); }

  private class EvenIter implements Iterator<T> {
    private int pos = 0;

    public boolean hasNext() { return pos < data.length; }

    public T next() {
      T next = data[pos]; pos += 2;
      return next;
    }
}}
```

Can access private fields/methods of enclosing class

Inner Classes: Example

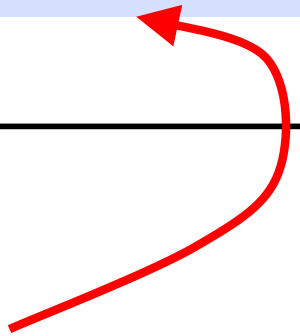
```
public class MyList<T> implemen
  private T[] data;

  public MyList(T[] d) { data = d; }

  public Iterator<T> iterator() { return new EvenIter(); }

  private class EvenIter implements Iterator<T> {
    private int pos = 0;

    public boolean hasNext() { return pos < data.length; }

    public T next() {
      T next = data[pos]; pos
      return next;
    }
}}
```

Enclosing class can construct and return instances of inner class

Other classes cannot construct instances as it's private

Inner Classes: External Construction

```
public class Shape {
  private class Square {
    private int x, y, width, height;
    private Square(int x, int y, int width, int height) { … }
  }

  public static void main(String[] args) {
    Shape parent = new Shape();
    Shape.Square square = parent.new Square(1,1,2,3);
  }
}
```



- Not static – needs an parent object!

Inner Classes: External Construction

```
public class Shape {
  private class Square {
    private int x, y, width, height;
    private Square(int x, int y, int width, int height) { … }
  }
}
```

```
public class ShapeTest {

  public static void main(String[] args) {
    Shape parent = new Shape();
    Shape.Square square = parent.new Square(1,1,2,3);
  }
}
```

- **Work or not?** **A) Yes** **B) No**

Inner Classes: External Construction

```
public class Shape {
  private class Square {
    private int x, y, width, height;
    private Square(int x, int y, int width, int height) { … }
  }
}
```

```
public class ShapeTest {

  public static void main(String[] args) {
    Shape parent = new Shape();
    Shape.Square square = parent.new Square(1,1,2,3);
  }
}
```

- **Work or not?**

A) Yes 

B) No 

Inner Classes: External Construction

```
public class Shape {
  public class Square {
    private int x, y, width, height;
    public Square(int x, int y, int width, int height) { … }
  }
}
```

```
public class ShapeTest {

  public static void main(String[] args) {
    Shape parent = new Shape();
    Shape.Square square = parent.new Square(1,1,2,3);
  }
}
```

- **Work or not?** **A) Yes** **B) No**

Inner Classes: External Construction

```
public class Shape {
  public class Square {
    private int x, y, width, height;
    public Square(int x, int y, int width, int height) { … }
  }
}
```

```
public class ShapeTest {

  public static void main(String[] args) {
    Shape parent = new Shape();
    Shape.Square square = parent.new Square(1,1,2,3);
  }
}
```

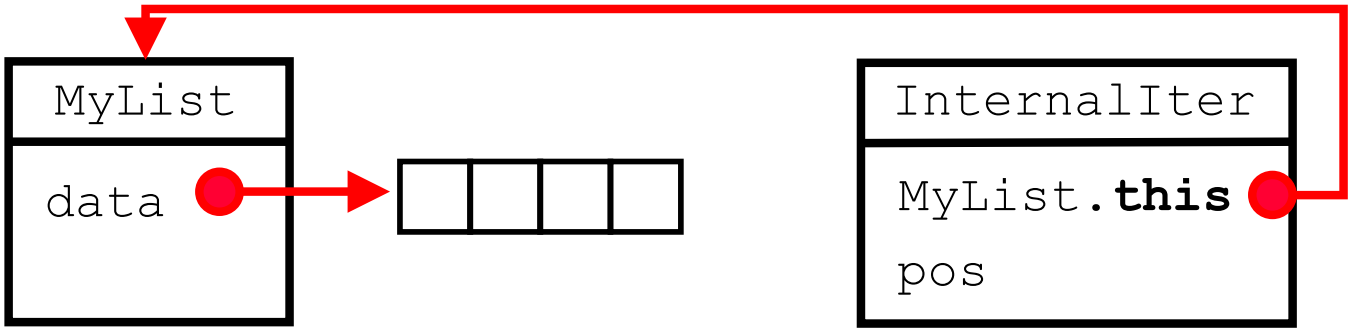
- **Work or not?**

A) Yes 

B) No 

Inner Classes: Scoping

- Inner classes have ***parent pointer***
  - For accessing fields/methods of enclosing class (parent)
  - Parent pointer automatically supplied for new inner class




```
public class MyList<T> implements Iterable<T> {
  private T[] data;

  private class InternalIter implements Iterator<T> {
    private int pos;
}}
```

Inner Classes: Scoping

```
public class MyList<T> implements Iterable<T> {
 private T[] data;

 public MyList(T[] d) { data = d; }

 public Iterator<T> iterator() { return new EvenIter(); }

 private class EvenIter implements Iterator<T> {
  private int pos = 0;

  public boolean hasNext() { return pos < data.length; }

  public T next() {
   T next = MyList.this.data[pos]; pos += 2;
   return next;
  }
}}
```

Explicit this-scoping

Static Inner Classes

```
public class Type {

  public static class IntType {
    …
  }

  public static class RealType {
    …
  }
}
```

- Static Inner Classes have **NO** parent pointer!
  - Can NOT access fields/methods of enclosing class
  - Can construct without providing parent pointer
  - If no need to access enclosing info, then this is more convenient (and potentially more efficient).

Which one(s) can work?

```
public class Outer {
  private class Inner { … }

  public static class StaticInner { … }

  public static void main (String[] args) {
    Inner c1 = new Outer.Inner();

    Inner c2 = new Outer().new Inner();

    StaticInner c3 = new Outer.StaticInner();

    StaticInner c4 = new Outer().new StaticInner();
  }
}
```

A

B

C

D

A B C D

Which one(s) can work?

A

B

C

D

```
public class Outer {
  private class Inner { … }

  public static class StaticInner { … }

  public static void main (String[] args) {
    Inner c1 = new Outer.Inner();

    Inner c2 = new Outer().new Inner();

    StaticInner c3 = new Outer.StaticInner();

    StaticInner c4 = new Outer().new StaticInner();
  }
}
```

A ✗ B ✓ C ✓ D ✗

Inner Class vs Static Inner Class

- Inner Class can access members of enclosing class, static cannot
- When being constructed, inner class needs to have a parent pointer, static does not need
- Inner class cannot have static methods, static can

Local Classes

- Sometimes we want to define classes that are only needed locally (e.g. within a block of a method)

```
class Outer {
  public Outer create(final int field) {
    class Inner extends Outer {
      private int myfield = field;
      …
    }
    …
    return new Inner();
  }
}
```

Local Classes

- Can be defined in any block
  - Method body
  - `for` loop
  - `if` clause
- Can access members of enclosing class
- Can access **`final`** local variables of enclosing block
- Can access effectively **`final`** local variables since Java 8 (never changed since initialisation)

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    final int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not?

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    final int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not?

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not?

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not?

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    number = 2;

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not?

Local Classes

```
public class LocalClassExample {
  public static void outMethod() {
    int number = 1;

    class Inner {
      public void printNumber() { System.out.println(number); }
    }

    number = 2;

    Inner c = new Inner();
    c.printNumber();
  }

  public static void main(String[] args) {
    outMethod();
}}
```

- In Java 8, work or not? 

Local Classes

- Cannot have static methods (same as Inner Classes)
- Must be non-static, so cannot declare interfaces as local classes
- Cannot have static member, unless it's final primitive (constant)

```
public void greetInEnglish(String name) {
   interface Greeting { public void greet(); }
   class EnglishGreeting implements Greeting {
     public static String prefix;
     public static final String HELLO = "Hello! ";
     public void greet() {
       System.out.println(HELLO + name);
}}}
```

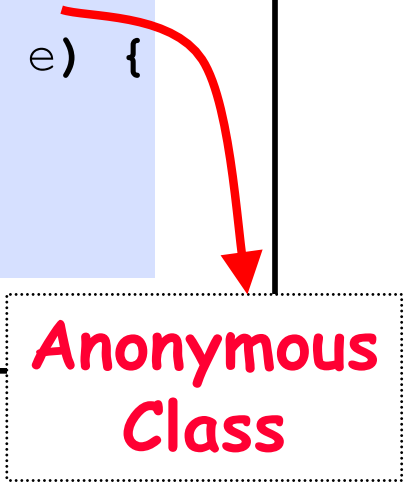
Anonymous Classes

- Make code more concise
- Declare and instantiate a class at the same time
- Local class, but with no name
- A typical example: **event handler**

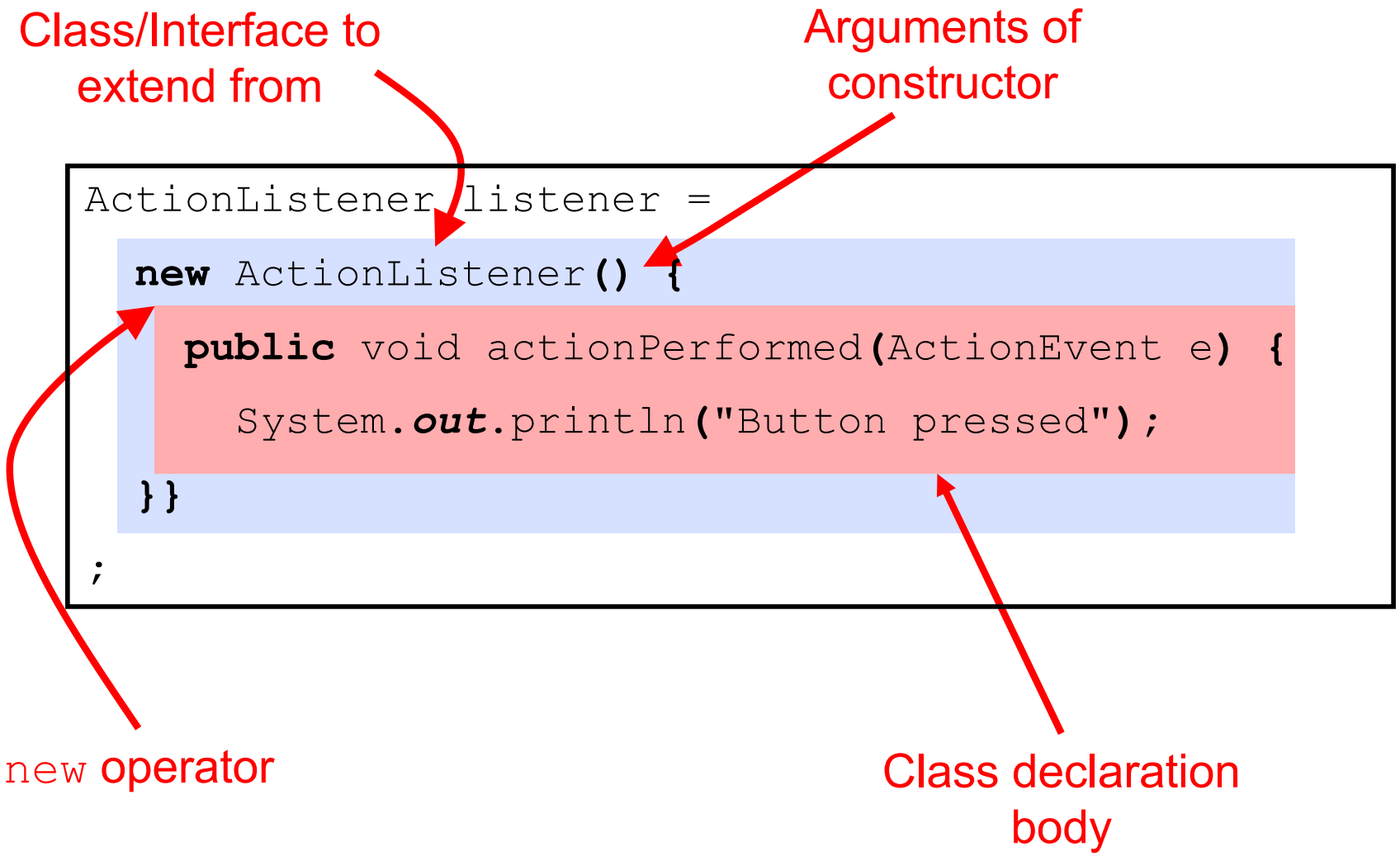
Event Handler

- The action listener of a button
- Basically each button will have its own listener
- Override the method `actionPerformed()`

```
JButton b = new JButton(”Press Me");
b.addActionListener(
  new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      System.out.println("Button pressed");
  }}
);
```



Anonymous Classes: Syntax



Anonymous Classes

- Like local classes
  - Access to enclosing class
  - Access to effectively final variables of enclosing scope
  - Cannot have static methods/interface
  - Static fields have to be constants
  - Can define inner/local classes inside
  - **Cannot declare constructor**

Anonymous Classes

```
ArrayList<String> ls = new ArrayList<String>();
…
Collections.sort(ls, new Comparator<String>(){
  public int compare(String s1, String s2) {
    return s1.compareToIgnoreCase(s2);
}});
```

Anonymous Classes

```
ArrayList<String> ls = new ArrayList<String>();
…
Collections.sort(ls, new Comparator<String>(){
  public int compare(String s1, String s2) {
    return s1.compareToIgnoreCase(s2);
}});
```

- Learn how to read the code through the syntax: fading away the anonymous class and method declaration, what you obtain read like:
- Sort `ls` using `s1.compareToIgnoreCase(s2)`

Summary

- Inner classes
  - Static vs Non-static
- Local classes
- Anonymous classes
- Relationship between enclosing class/block/scope
- Restrictions