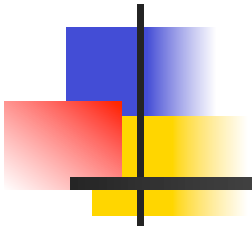
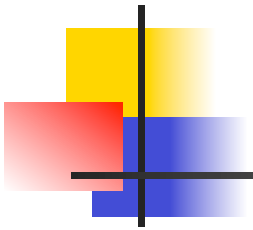


Anonymous Inner Classes and Task Decomposition

Corky Cartwright
Vivek Sarkar
Department of Computer Science
Rice University



Acknowledgments

CISC370 course taught by Chris Fischer, U.Delaware (Lecture 9)

http://www.cis.udel.edu/~cfischer/cisc370/

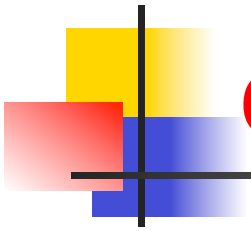
“Introduction to Concurrent Programming in Java”, Joe Bowbeer, David Holmes, OOPSLA 2007 tutorial slides

Contributing authors: Doug Lea, Brian Goetz

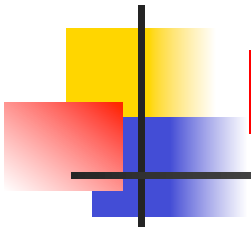
Code Example: BiIterator class from BiList.java (HW 10)

```
private class BiIterator implements BiIteratorI<T> {
   Node<T> current;
   BiIterator() {
      current = BiList.this.head.succ;  // current is first item (if it exists)
   }
   public void first() {
      current = BiList.this.head.succ;  // current is first item (if it exists)
   }
   public void last() {
      current = BiList.this.head.pred;  // current is last item (if it exists)
   }

   public void next() { current = current.succ; }    // wraps around end
   public void prev() { current = current.pred; }    // wraps around end
```

Code Example: BiIterator class from BiList.java (cont.)

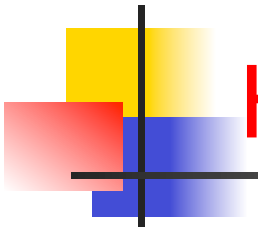


```
    public T currentItem() {
        if (current == BiList.this.head) {
            throw new
                IteratorException("No current element in " + BiList.this);
        }
        return current.item;
    }

    public boolean atEnd() { return current == BiList.this.head; }
} // BiIterator
```



Inner Classes

- An Inner class is a class defined inside of another class.
- Why would anyone want to do that?
  - An Inner class object can access the implementation of the object that created it – including private fields
  - Inner classes can be hidden from other classes in the same package
  - Anonymous inner classes are frequently used for creating event callbacks, and for task decomposition in parallel programming



How to declare a named inner class

- You declare a named inner class like any other member

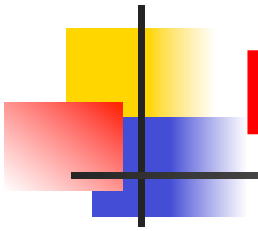
```
class ICExample {
   private class thisIsAnInnerClass { //THIS IS THE INNER CLAS
       // define a class here
   }
}
```

Referring to members of Outer Classes

- With inner classes, you can refer to members of an outer class using the outer class' name (if necessary for disambiguation)

```
public void actionPerformed(ActionEvent e) {
   // inside the inner class
   double interest = <OuterClassName>.balance * this.interestRate;
   <OuterClassName>.balance += interest;
}
```

- The balance field is a private member of the outer class.



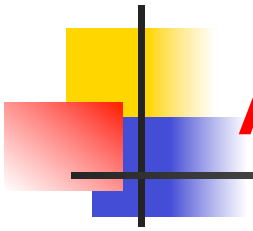
Local Inner classes

- You can declare an inner class inside of a method, just like you could declare a local variable.
- A local inner class can refer to final members of the enclosing class, and to final local variables in the enclosing method
- When using a local inner class, if you only want to make one instance of it, you don't even need to give it a name
  - This is known as an *anonymous inner class*
- These are convenient for event programming and parallel programming
- However, the syntax is extremely cryptic …

Anonymous Inner classes

```
public void start(final double rate)
{
    ActionListener adder = new
            ActionListener()
    {
            public void actionPerformed(ActionEvent evt)
            {
                    double interest = balance * rate / 100;
                    balance += interest;
            }
    };
    Timer t = new Timer(1000, adder);
    t.start();
     . . .
}
```

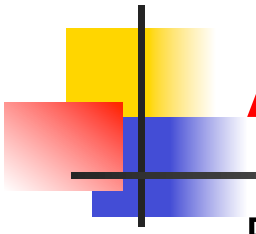
- This is saying, construct a new object of a class that implements the ActionListener interface, where the one required method (actionPerformed) is defined inside the brackets.



Anonymous Inner classes

- You have to look very carefully to see a difference between construction of a new object, and construction of a new inner class extending a class.

```
//A person object
Person queen=new Person(“Mary”);  //Person Object
//An object of an inner class extending Person
Person count = new Person(“Frankenstein”) {  //class code here};
```



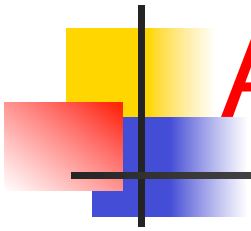
Anonymous Classes

- For each function that we want to use as a value, we must define a class, preferably a singleton. Since the class has no fields, all instances are effectively identical.

- Java provides a lightweight notation for singleton classes called *anonymous classes*. Moreover these classes can refer to fields and `final` method variables that are in scope.

- Anonymous class notation:

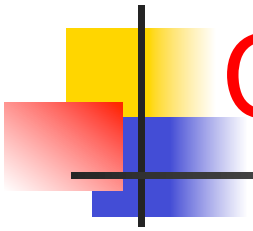
```
new <type>() {
  <member1>
  ...
  <membern>
}
```



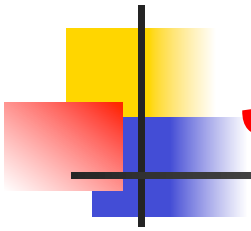
Anonymous Class Example

```
. . .
final Integer negativeOne = new Integer(-1);
ObjectList ol1 = . . .;
ObjectList ol2 = ol1.map(
    new UnaryFun() { // Anonymous inner class
     Object apply(Object arg) {
          if (arg.predicate())
          return EmptyObjectList.ONLY.cons(arg);
          else
            return negativeOne; // Free variable
       }
  }
);
```

Free Variables in Anonymous Classes



- What do free variables mean inside anonymous classes? What do they mean in λ-expressions?
- In Java, the free variables can be either:
  - fields, or
  - local (method) variables.



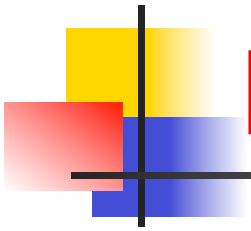
Java's Callable Interface

- Introduced in J2SE 5.0 in java.util.concurrent package (remember to "import java.util.concurrent;")

```
public interface Callable<V> {
      /**
       * Computes a result, or throws an exception.
       *
       * @return computed result
       * @throws Exception if unable to compute a result
       */
      V call( ) throws Exception;
}
```

Task Decomposition using Callable

```
// HTML renderer before decomposition
   ImageData image1 = imageInfo.downloadImage(1);
   ImageData image2 = imageInfo.downloadImage(2);
   . . .
   renderImage(image1);
   renderImage(image2);

// HTML renderer after task decomposition
   Callable<ImageData> task1 = new Callable<ImageData>() {
     public ImageData call() {return imageInfo.downloadImage(1);}};
   Callable<ImageData> task2 = new Callable<ImageData>() {
     public ImageData call() {return imageInfo.downloadImage(2);}};
   . . .
   renderImage(task1.call());
   renderImage(task2.call());
```



Food for thought

- What have we achieved by replacing “renderImage(image1);” by “renderImage(task1.call());” ?

- When is it legal to perform this kind of substitution in a program? When is it not?