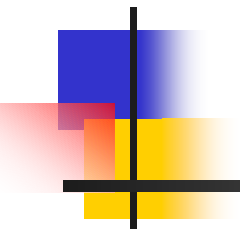
Java Programming
Day 2

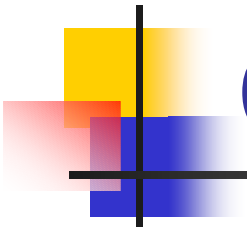


Robert Chatley

rbc@doc.ic.ac.uk

William Lee

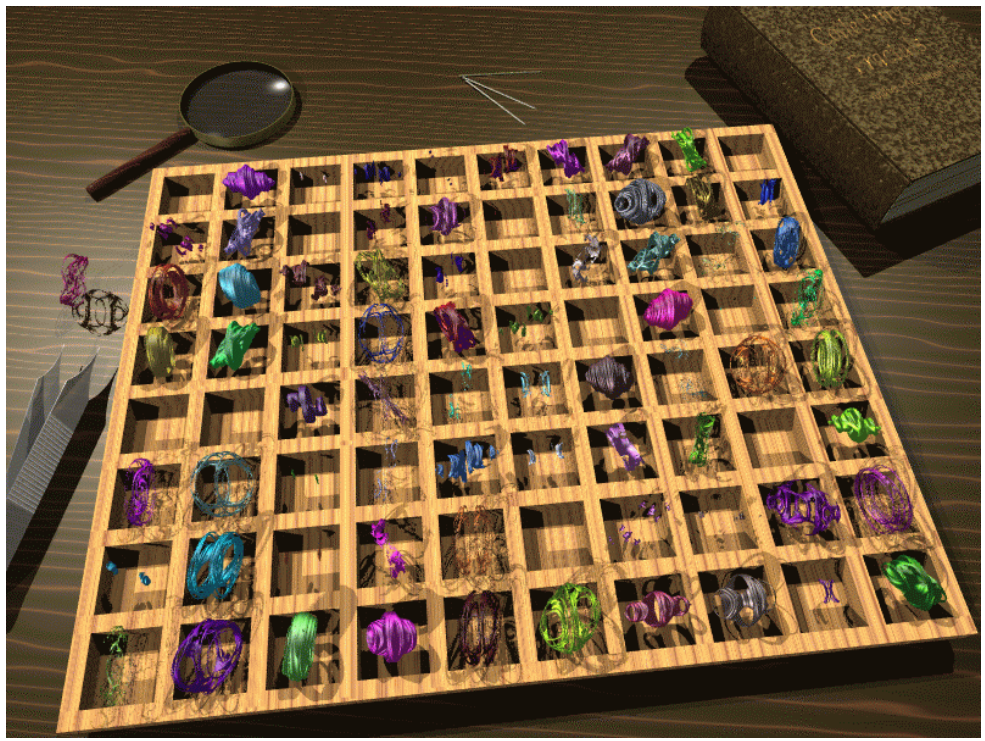
wwhl@doc.ic.ac.uk

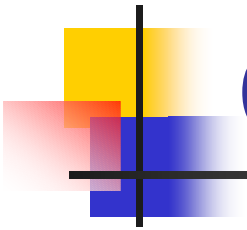


Course Outline

- Day 1 - Introduction
  - Java VM and Memory Model
  - Java Object-Orientation
  - Key language features
- Day 2 - The standard library
  - Collections and ADTs
  - Input/Output
  - GUI programming with Swing
  - Testing

Collections

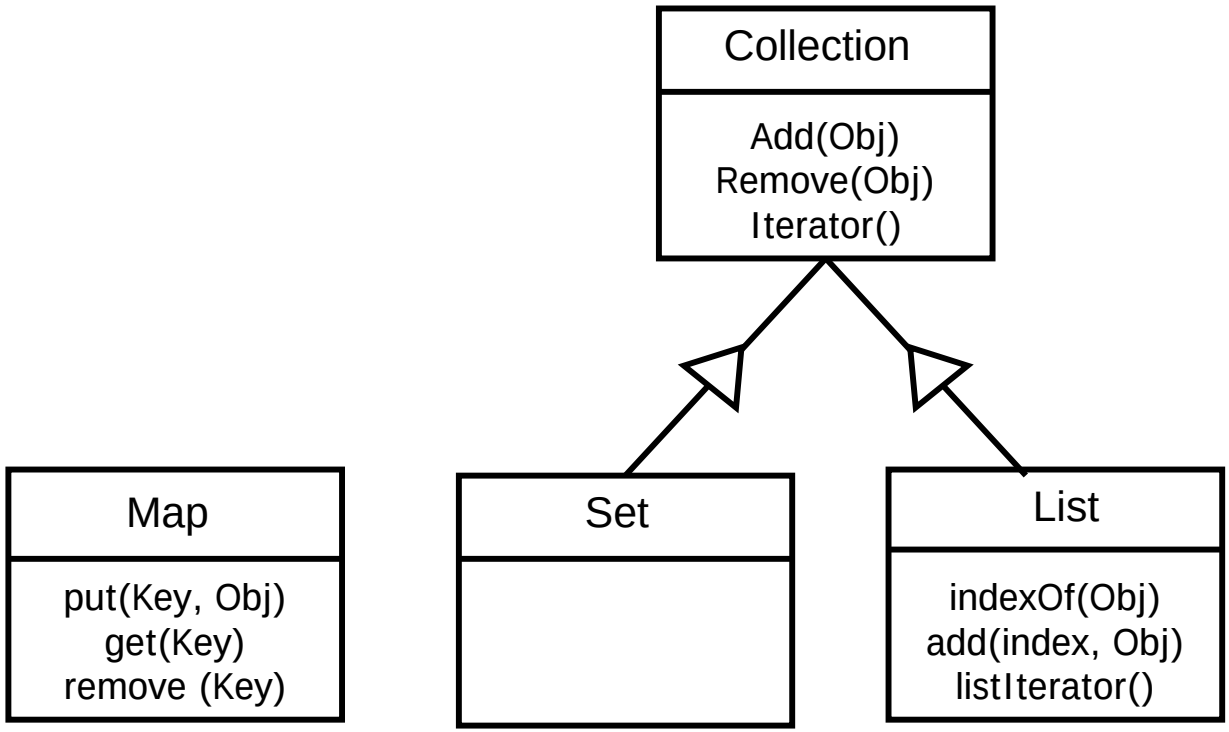




Collections

- Abstract Data Type
- Collection Interface
  - java.util.List - ordered items
  - java.util.Map - indexable items
  - java.util.Set - unordered non-duplicate items

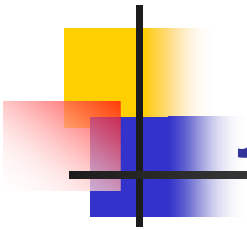
Collections API



Casting

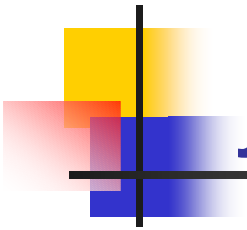
```
import java.util.*;

public class ListExample {

  void go() {

    List team = new L

    team.add( new Pla
    team.add( new Player
    Player p = (Player)team.get(0);
    p.play();
  }
}
```

We know the list contains Players so we can *cast* the result



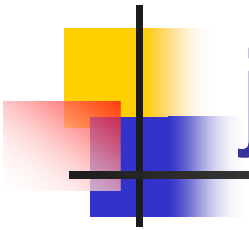
java.util.Set

- Implementations
  - HashSet - (Constant time)
  - SortedSet
    - TreeSet - (log(n) time)



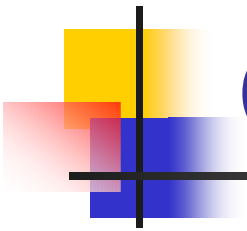
java.util.List

- Implementations
  - ArrayList
  - LinkedList
  - Vector (synchronized)
    - Stack - LIFO



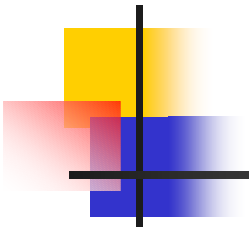
java.util.Map

- Implementations
  - HashMap
  - HashTable (synchronized)
  - SortedMap (ordered by key order)
    - TreeMap (Red-black tree, log(n))
  - WeakHashMap (entry can be garbage collected when key is not in use)



Other Collection Classes

- java.util.Collections
  - Helper methods working on Collection classes
- java.util.Arrays
  - Helper methods working on java array (Object[])



Iterators

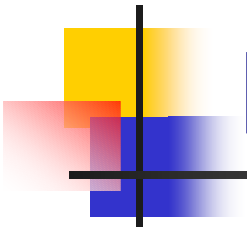
- Commonly used pattern for processing each element of a collection in turn
- The Iterator captures a snapshot of the collection an allows you to step through the data

Iterators

```
import java.util.*;

…
 List team = new LinkedList();

 team.add( new Player( “Beckham” ));
 team.add( new Player( “Owen” ));

 for(Iterator i = team.iterator() ; i.hasNext() ; ) {

   Player p = (Player)i.next();
 }
```

We get an iterator from the List

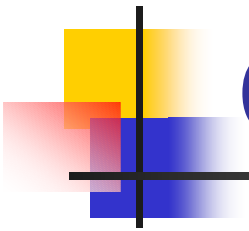
The list contains Players so the Iterator contains Players - we can cast the result



New in Java 1.5 - Generics

- Rather than all collections being of Objects, with *generics* we can specify Lists of Strings or Sets of Dogs...
- Removes need for casting

- JDK 1.5 is available from http://java.sun.com



Generic Lists

```
import java.util.List;

public class GenericListExample {

  void go() {

    List<Player> team = new LinkedList<Player>();

    team.add( new Player( “Beckham” ));
    team.add( new Player( “Owen” ));

    Player p = team.get( 0 );
    p.play();
  }
}
```

Generic Iterators

```
import java.util.List;

…

List<Player> team = new LinkedList<Player>();

team.add( new Player( “Beckham” ));
team.add( new Player( “Owen” ));

for( Iterator<Player> i = team.iterator() ;
                                  i.hasNext() ; ) {

   Player p = i.next();
   p.play();
}
```

Foreach Loops

- A more concise way...

```
import java.util.List;

…

List<Player> team = new LinkedList<Player>();

team.add( new Player( “Beckham” ));
team.add( new Player( “Owen” ));

for ( Player p : team ){

   p.play();
}
```

For each Player p in team...

Generic Classes

```
public class Pair<A,B> {

  private A firstElement;
  private B secondElement;

  public Pair( A fst, B snd ) {

    firstElement = fst;
    secondElement = snd;
  }

  public A first()  { return

  public B second() { return

}
```

A and B are *type parameters* which are filled in when an object is created

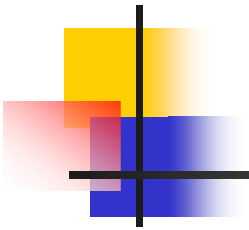
We can use A and B as types throughout the class definition

Using Generic Types

```
public class Pair<A,B> {
  // as before
}
class PairTest {
  void go() {
    Pair<Dog,Cat> pets =
              new Pair<Dog,Cat>(new Dog(),new Cat());
  }
}
```

A and B are substituted for these types for this instance of Pair

Input/Output (java.io)



Input/Output (java.io)

- Data I/O Abstraction is provided through the concept of input & output streams.
- Application works with the stream abstraction regardless of the type of the source or sink of data.
- java.io.InputStream & java.io.OutputStream
  - byte - raw binary data
- java.io.Reader & java.io.Writer
  - char - textual data - one or more bytes depending on the encoding (e.g. UNICODE)
  - perform encoding/decoding of char-to-byte(s) with a given character set or use the platform default

Reading a Character File

```
File myFile = new File(
Reader input =
try {
  input = new FileReader( myFile );
  int charRead = 0;
  char buffer[] = new char[1024];
  while((charRead = input.read(buffer, 0, 1024)) > 0){
        // process your input here
        System.out.print(buffer);
  }
} catch ( IOException ioe ) {
    System.err.println( "Error reading stream
} finally {
  try { if ( input != null ) input.close(); }
  catch ( IOException ioe ) {
    System.err.println( "Error closing s
  }
}
```

input is of type Reader

input is assigned a FileReader instance. Polymorphism!

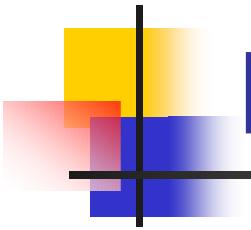
Variables to keep track of the number of character read and a buffer for subsequent read operation

Keep reading until charRead <= 0. ***buffer*** contains characters read from the Reader. It might not be completely filled! Check ***charRead***.

Always clean-up after you.

Reading a Binary File

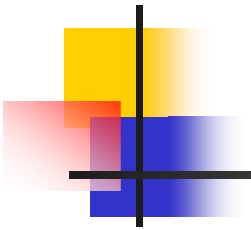
```
File myFile = new File("/tmp/myfile.txt");
InputStream input = null;

try {

  input = new FileInputStream( myFile );

  int byteRead = 0;

  byte buffer[] = new byte[1024];

  while((byteRead = input.read(buffer)) > 0){

        // process your input here

        System.out.write(buffer, 0, byteRead);

  }

} catch ( IOException ioe ) {
    System.err.println( "Error reading stream" );
} finally {

  try { if ( input != null ) input.close(); }
  catch ( IOException ioe ) {
    System.err.println( "Error closing stream" );
  }
}
```



Boiler Plate

```
InputStream input = null;
try {
  input = … construct InputStream implementation …
  int byteRead = 0;
  byte buffer[] = new byte[1024];
  while((byteRead = input.read(buffer)) > 0){
        // process your input here
        …
  }
} catch ( IOException ioe ) {
    System.err.println( “Error reading stream” );
} finally {
  try { if ( fr != null ) fr.close(); }
  catch ( IOException ioe ) {
    System.err.println( “Error closing stream” );
  }
}
```

InputStream (Reader) Implementations



- java.io.FileInputStream (java.io.FileReader)
  - Local file as source
- java.io.ByteArrayInputStream (java.io.CharArrayReader)
  - byte[] as source
- java.io.BufferedInputStream (java.io.BufferedReader)
  - Another InputStream as source
  - Buffer read operation issued to the underlying InputStream. Reduce number of “int read()” call to slow peripheral.
- java.io.DataInputStream (n/a)
  - Another InputStream as source
  - Reads java primitive in a portable manner. Input must be previously produced by a java.io.DataOutputStream.
- java.io.ObjectInputStream (n/a)
  - Another InputStream as source
  - Reads “serializable” objects from stream. Input must conform to the Java Object Serialization Specification, or produced by a java.io.ObjectOutputStream.
- Many more...

Writing to a Character File

```
File myFile = new File("/tmp/myfile.txt");
Writer output = null;
try {
  output = new FileWriter( myFile );
  output.write("hello world");
  output.write(buffer, 0, buffer.length);
} catch ( IOException ioe ) {
    System.err.println( "Error writing strea
} finally {
  try { if ( output != null ) output.close(
  catch ( IOException ioe ) {
    System.err.println( "Error closing strea
  }
}
```

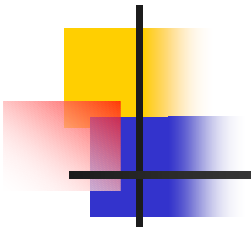
output is a Writer

output is instantiated as a FileWriter

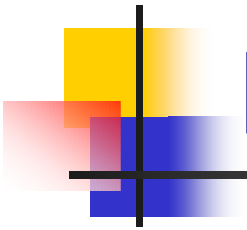
Use write() operation to write String, char[] or integer to the output.

Always clean-up after you.

OutputStream (Writer) Implementations



- java.io.FileOutputStream (java.io.FileWriter)
  - Local file as sink
- java.io.ByteArrayOutputStream (java.io.CharArrayWriter)
  - byte[] can be obtained from toByteArray()
- n/a (java.io.StringWriter)
  - Accumulates write() to produce a String
- n/a (java.io.OutputStreamWriter)
  - Wraps an OutputStream as a Writer
  - java.io.InputStreamReader counterpart
- java.io.BufferedOutputStream (java.io.BufferedWriter)
  - Another OutputStream as sink
  - Buffer write operation issued to the underlying OutputStream. Reduce number of “void write(int)” call to slow peripheral.
- java.io.DataOutputStream (n/a)
  - java.io.DataInputStream counterpart
- java.io.ObjectOutputStream (n/a)
  - java.io.ObjectnputStream counterpar
- Many more...



Network Access

- java.net.* provides socket abstraction for TCP/UDP client/server low-level access.
- java.net.URL provides higher-level abstraction to some well-known protocols (e.g. HTTP(s), FTP)

Client Socket

Reading DoC home page using the HTTP protocol

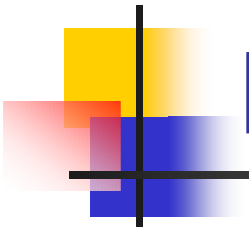
```
Socket socket = null;
try {
  socket = new Socket("www.doc.ic.ac.uk", 80);
  OutputStream os = socket.getOutputStream();
  InputStream is = socket.getInputStream();
  os.write("GET /index.html\n");
  int byteRead = 0;
  byte buffer[] = new byte[1024];
  while((byteRead = is.read(buffer)) > 0){
          System.out.write(buffer, 0, byteRead);
  }
} catch ( IOException ioe ) {
    System.err.println( "Error writing stream" );
} catch ( UnknownHostException ioe ) {
    System.err.println( "Host not found" );
} finally {
  try { if ( socket != null ) socket.close(); }
  catch ( IOException ioe ) {
    System.err.println( "Error closing socket" );
  }
}
```

Server Socket

Single-threaded “Echo” Server

```
ServerSocket socket = null;
try {
 socket = new ServerSocket(8888);
 while(true){
   Socket clientSocket = null;
   try {
     // wait for new connection
     clientSocket = socket.accept();
     // new connection accepted
     BufferedReader reader = new BufferedReader(
       new InputStreamReader(clientSocket.getInputStream()));
     String line = reader.readLine();
     if(line != null){
       clientSocket.getOutputStream().write(line.getBytes());
     }
   } catch (IOException xEx) {
     xEx.printStackTrace();
   } finally {
     if(clientSocket != null) {
       try {
         clientSocket.close();
       } catch (IOException xEx) {
         xEx.printStackTrace();
       }
     }
   }
 }
} catch ( IOException ioe ) {
 ioe.printStackTrace();
} finally {
 if(socket != null) {
   try {
     socket.close();
   } catch (IOException xEx) {
     xEx.printStackTrace();
   }
 }
}
```

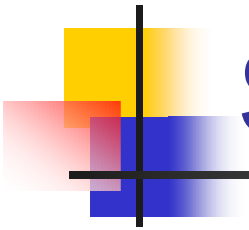
Can be made multi-threaded to allow server to accept concurrent connections



High-level Access

- java.net.URL provides read access to popular protocol (e.g. http://, ftp://, file://, myown://)
- java.rmi.*: Consider Java Remote Method Invocation (RMI) for transparent remote object invocation
- Web Services: Standardised protocols for invoking message-based or rpc-based remote services.

Swing

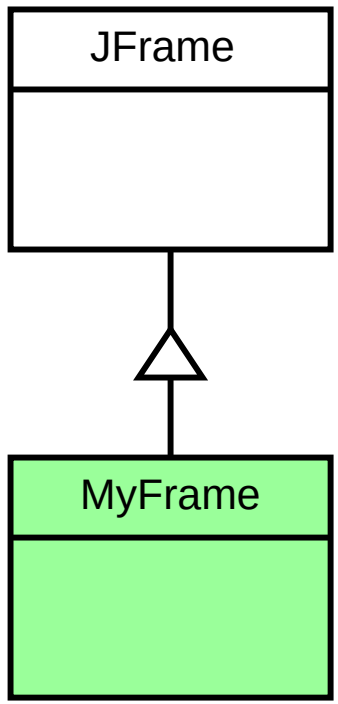


Swing

- Java's GUI toolkit
- Easy to make programs with windows, buttons, menus, etc
- Swing classes live in **`javax.swing`**
  Most begin with a J (JButton, JMenu...)

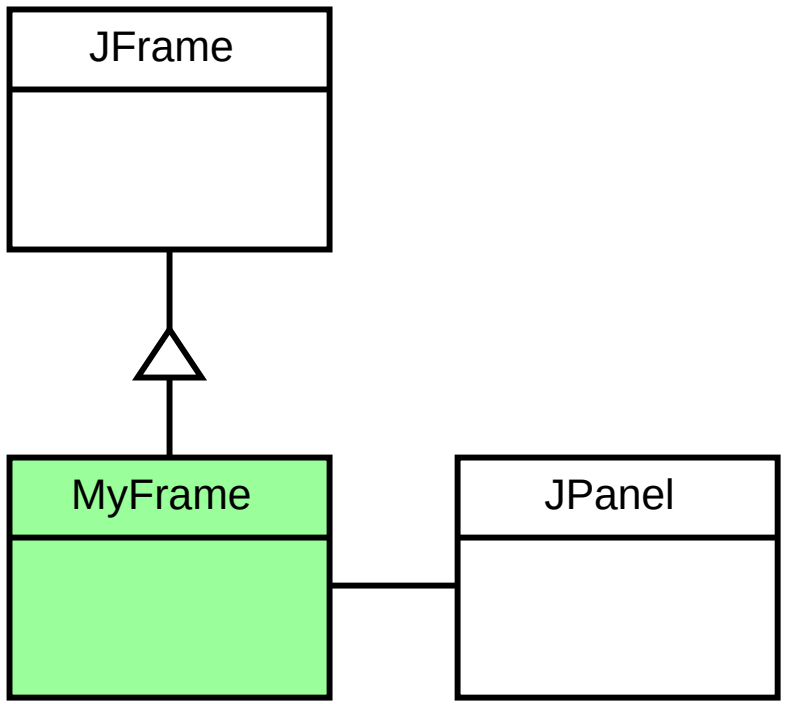
Starting off - JFrame

- JFrame provides basic window functionality
- Make your main GUI class extend JFrame
- Use **`pack()`** and **`show()`**



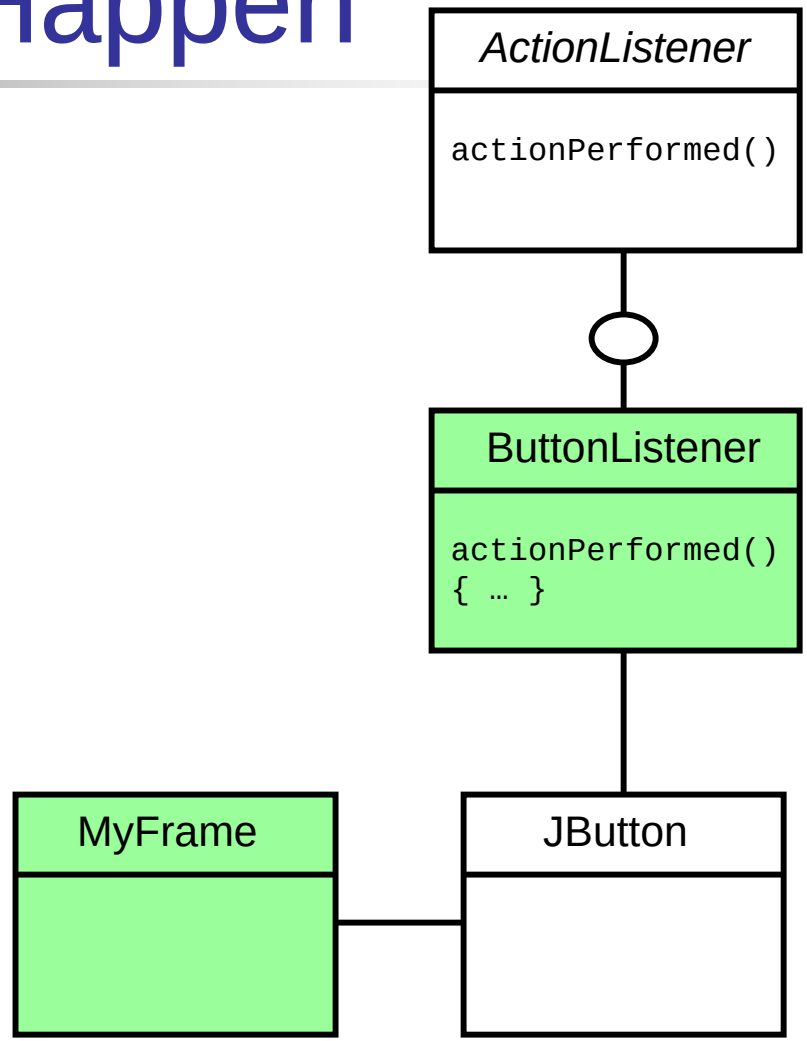
Adding things to your frame

- Frames don't do much on their own
- Need to put something in it to display text/graphics
- Add a JTextArea or JPanel



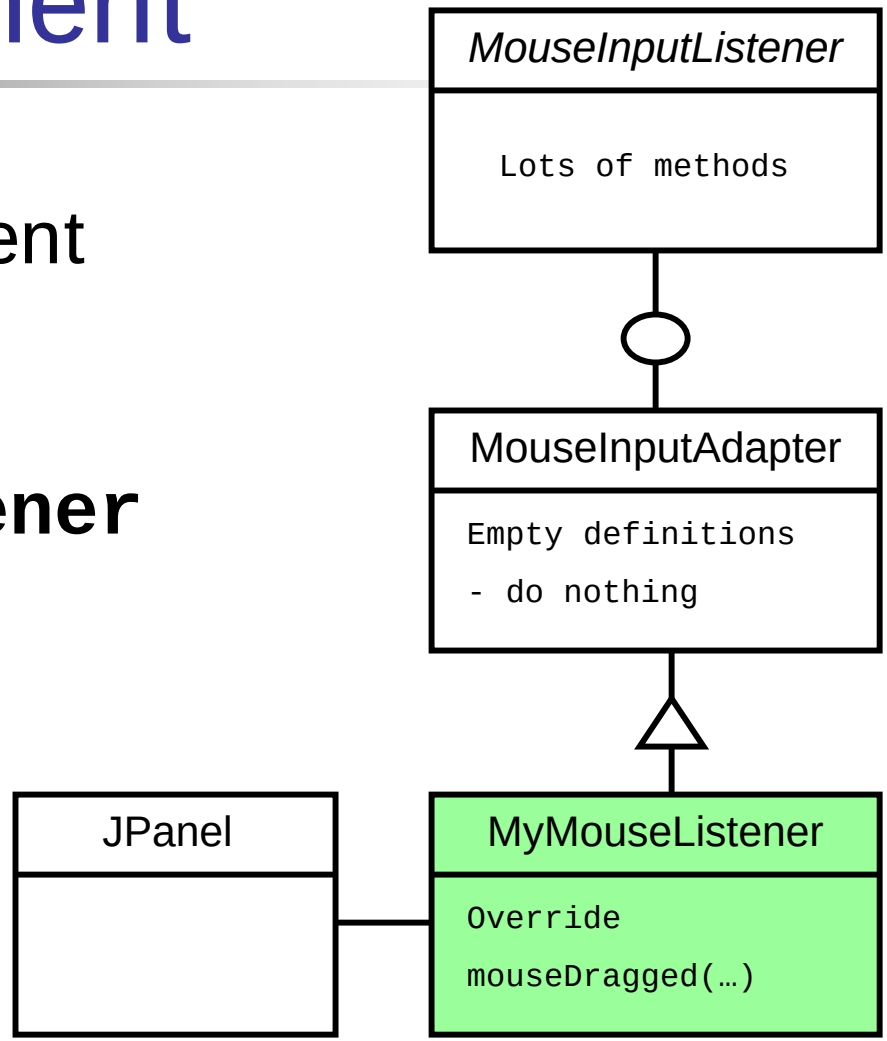
Making things Happen

- Listen for events (button presses etc)
- Swing will call methods you define when events occur
- **`ActionListener`** interface



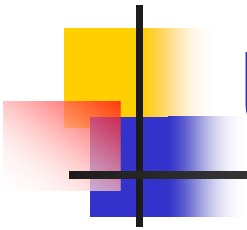
Mouse Movement

- Different type of event **`MouseEvent`**
- **`MouseInputListener`** interface
- Use an adapter to save code



Testing


WE STILL HAVE TOO MANY SOFTWARE FAULTS. WE'LL MISS OUR SHIP DATE.
MOVE THE LIST OF FAULTS TO THE "FUTURE DEVELOPMENT" COLUMN AND SHIP IT.
90% OF THIS JOB IS FIGURING OUT WHAT TO CALL STUFF.
www.dilbert.com scottadams@aol.com
© 2004 Scott Adams, Inc./Dist. by UFS, Inc.
© UFS, Inc.

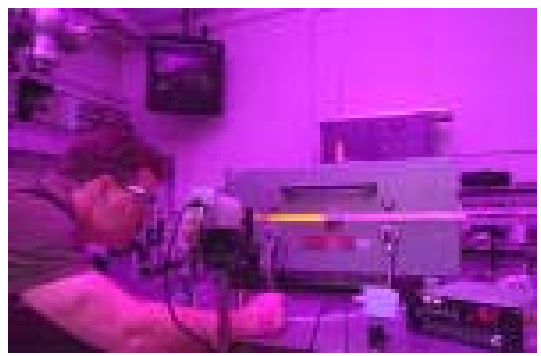


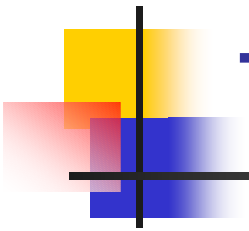
Unit Testing

- Not a feature of the language, but something that is useful
- Write tests for each class as you develop your program
  - You know that things work
  - It tells you when things break

JUnit

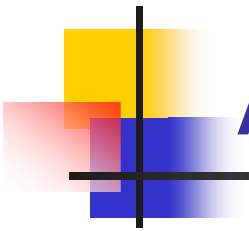
- JUnit library supports unit testing for Java
- Integrated with Eclipse IDE
- Very quick and easy to create and run tests
- www.junit.org





Test Cases

```
import junit.framework.TestCase;
import java.util.*;

public class MapTest extends TestCase {

  public MapTest(String arg0) { super(arg0); }

  public void testWhatGoesInComesOut() {
    Map x_map = new HashMap();
    String x_key = "key";
    String x_val = "some data";
    x_map.put( x_key , x_val );
    assertSame( x_val , x_map.get( x_key ) );
  }
}
```



A Failing Test

```
import junit.framework.TestCase;
import java.util.*;

public class MapTest extends TestCase {

  public MapTest(String arg0) { super(arg0); }

  public void testWhatGoesInComesOut() {
    Map x_map = new HashMap();
    String x_key = “key”;
    String x_val = “some data”;
    x_map.put( x_key , x_val );
    x_map.put( x_key , “something new” );
    assertSame( x_val , x_map.get( x_key ) );
  }
}
```

Running All Your Tests

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

  public static Test suite() {

    TestSuite suite = new TestSuite("all tests");

    suite.addTest( new TestSuite(ListTest.class));

    suite.addTest( new TestSuite(MapTest.class));

    return suite;
  }
}
```