CS 11 java track: lecture 4

- This week:
  - arrays
  - interfaces
  - listener classes
  - inner classes
  - GUI callbacks

arrays (1)

- array: linear sequence of values
- arrays are real objects in java
  - have one public field: `length`
  - are created ONLY using new:

```
// Create an array of ten ints:
int[] arr = new int[10];
int arr[10];  // NOT VALID!
```

arrays (2)

- can't change dimensions of array after creation!
- can declare in either of two ways:

```
int[] arr;
int arr[];
```

- first way is better (more obvious)
- array variable `arr` is `null` until assigned

arrays (3)

- what are the initial contents of an array?

```
int[] arr = new int[10];  // contents?
```

- answer: default value for type:
  - int: 0
  - float, double: 0.0
  - char: `'\0'`
  - boolean: **`false`**
  - Object: **`null`**

arrays (4)

```
String[] arr = new String[10];
```

- What is `arr[0]`?
  - `null`
  - `String` is an `Object`

multidimensional arrays

- can have multidimensional arrays (arrays of arrays):

```
int[][] arr = new int[3][2];
```

- initial contents of **`arr`**?

array initialization quiz (1)

```
// What do the following variables contain
// after these lines execute?
int[] x;
int[][] y;
int[] x = new int[3];
int[][] y = new int[3][2];
int[][] z = new int[3][]; // ???
for (int i = 0; i < 3; i++) {
    z[i] = new int[2];
}
```

array initialization quiz (2)

```
// What do the following variables contain
// after these lines execute?
Object[] w;
w = new Object[10];
w[0] = new Object();
```

explicit array initialization (1)

```
int[] x = new int[3];  // OK
int[] x = new int[] { 1, 2, 3 }; // OK
int[] x = new int[3] { 1, 2, 3 };  // INVALID!
// DUMB!!!
Object[] o = new Object[] {
    new Object(), new Object(), new Object()
};
int[] x;
x = new int[] { 1, 2, 3 }; // OK
x = new int[] { 4, 5, 6 }; // also OK
```

explicit array initialization (2)

```
int[][] y = new int[3][2];  // OK
int[][] y = new int[][] {
  { 1, 2 }, { 3, 4 }, { 5, 6 }
}; // OK
int[][] y = new int[3][2] {
  { 1, 2 }, { 3, 4 }, { 5, 6 }
}; // INVALID again!
```

- must specify dimensions in **`new`** stmt or else initialize – not both!

interfaces

- saw interfaces last time
- specify *behavior only*
  - no method bodies (just signatures)
  - no fields
- classes *implement* interfaces
  - then can treat instance of class as if it were an instance of the interface
  - can implement any number of interfaces

Listener classes (1)

- a class that "listens" for an event is a listener class
- listeners usually specified with interfaces:

```
public interface ActionListener {
    public void ActionPerformed(ActionEvent e);
}
```

- class that implements this can "listen" for actions of some kind and respond to them
- **`ActionEvent e`** contains more info about event
    - usually don't need it

Listener classes (2)

```
import java.awt.event.*;
public class MyWidget implements ActionListener
{
    // ... other code ...
    public void ActionPerformed(ActionEvent e)
    {
        // implementation of listener action
    }
    // ... other code ...
}
```

Listener classes (3)

- what an "Action" is depends on the class
  - a button → clicking on the button with the mouse
  - **`ActionListener`** used when only one action is meaningful
- for more elaborate kinds of listening, need more sophisticated listeners
  - e.g. **`MouseListener`**
  - much more complex

Listener classes (4)

```
public interface MouseListener {
    public void mouseClicked(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
}
```

- see? ☺
- for mouse movement (*e.g.* dragging) need **`MouseMotionListener`**
- **`MouseEvent e`** includes getX(), getY() methods etc.

making Listener classes

- three ways to make an instance of a listener
  - use **`ActionListener`** as a simple example
  - 1. make your class implement **`ActionListener`** directly
  - 2. create an inner class that implements **`ActionListener`**
  - 3. create an anonymous inner class that implements **`ActionListener`**

making Listener classes – way 1

- make your class implement **`ActionListener`** directly

```
public class MyWidget implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // do stuff...
        // ActionEvent e usually not needed
    }
    public MyWidget() {
        // initialize...
        addActionListener(this);
    }
}
```

making Listener classes – way 1

- NOTE: for this to work your class has to define the method **`addActionListener()`**
- *e.g.* **`JButton`** class does this
- in this case, the class itself is acting as its own **`actionListener`**
  - i.e. it handles the responsibility itself

making Listener classes – way 2

- can define an `actionListener` as an inner class (class defined inside another class)
- inner classes have access to surrounding class' private fields and methods
- can *e.g.* change surrounding class field values from method in inner class

making Listener classes – way 2

```
public class MyWidget {
    class MyListener implements ActionListener
    {
        public void ActionPerformed(ActionEvent e) {
            // do stuff
        }
    }
    public MyWidget() {
        // other initializations...
        addActionListener(new MyListener());
    }
}
```

making Listener classes – way 3

- sometimes creating an inner class just for one method feels like too much work
  - *e.g.* when only one instance will ever be created
- java provides a shortcut: anonymous inner classes
  - classes without constructors
  - usually no fields either; just methods
  - "lightweight" classes

making Listener classes – way 3

```
public class MyWidget {
    public MyWidget()
    {
        // some initializations...
        addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // do stuff...
            }
        });
    }
    // fields, other methods, etc.
}
```

making Listener classes – way 3

- anonymous inner class is a class created "on-the-fly"
- saves boring typing (no `class` declaration)
- not as flexible as inner classes (no constructor)
- usually the right solution for listeners

next week

- the `final` keyword
- introduction to threads
- design advice