Chapter 7: Pinball

Game Construction Kit

Vectors

- Example of a "collection class"
- Must "import java.util.Vector"
- More flexible than arrays: will grow
- Define: private Vector balls;
- Init: balls = new Vector();
- Add an entry: balls.addElement(b1);
- Access: balls.elementAt(i);
- How many entries: balls.size();

Vectors (contd)

- Can only hold objects, not primitives:
  int I = 17;
  Balls.addElement(i); // NOT POSSIBLE
  Balls.addElement(new Integer(i)); // OK
- Internally: uses Object[] array, therefore must cast elements after retrieval:
  PinBall aBall = (PinBall) balls.elementAt(i);

MouseListener Interface

```
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);
}
```

Adapter class (library supplied)

```
public class MouseAdapter implements MouseListener {
    public void mouseClicked(MouseEvent e) {}
    … }
```

- All five methods don't do anything, but we can define a subclass overriding just those we really want:

```
public class MouseKeeper extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
     … // some code
     }}
```

More on MouseKeeper

- So MouseKeeper is a subclass of MouseAdapter, therefore implements the interface MouseListener, so we can:
  addMouseListener(new MouseKeeper());
- MouseEvent: provides useful methods like getX() and getY() to determine the mouse position at the occurrence of the event: e.g. mouseReleased() => release position

Multiple Threads of Execution

- Previous chapters: animation was a loop inside paint(g) method, now we explicitly create a separate thread
- A thread is like a parallel process, but inside the same memory address space
- Creation: subclassing builtin Thread class

PinBallThread extends Thread

```
{ private Ball = theBall;
  public PinBallThread (Ball aBall) {  // CONSTRUCTOR
     theBall = aBall;}
  public void run() {  // RUN METHOD
    while (theBall.y() < FrameHeight) {
      theBall.move();
      repaint();
      try { sleep(10);
      } catch (InterruptedException e) {System.exit(0);
}}}
```

How to create

- Must generate Ball and Thread, and start explicitly:

```
PinBall newBall = new PinBall(x,y);
balls.addElement(newBall);
Thread newThread = new
    PinBallThread(newBall);
newThread.start();
```

- Don't call run() directly.

Exception handling

```
try { sleep(10);
     } catch (InterruptedException e)
  {System.exit(0); }
```

- An Exception:an error condition that we can catch and handle systematically
- If some method (like sleep()) can throw an exception, we need to wrap it inside a try block (or declare that our method might throw on this exception; see the GuessingGame IO for another example)

PinBallTarget interface

- Want a uniform way of handling all the various targets
- Same behavior, but different structures:

```
interface PinBallTarget {
    public boolean intersects (Ball aBall);
    public void moveTo(int x, int y);
    public void paint(Graphics g);
    public void hitBy(Ball aBall);
    }
```

Various targets

- Both Balls and Targets use Rectangle, which defines an intersects() method
- See how Spring, Wall, and Hole implement the interface PinBallTarget
- But Hole also extends Ball, and ScorePad extends Hole, overriding e.g. paint() and hitBy()
- Structural relationship => behavioral relationship, but not always vice versa;
- if just behavioral relationship: use interface

Class Label

- Another AWT supplied class:
  - scoreLabel = New label("Score = 0");
  - scoreLabel.setText("Score = " + score);
- What if two (or more) threads want to update the score simultanously?
- Must ensure update happens as one single step (socalled "atomic action")
- One Java way: synchronized methods

Synchronized methods

- Never executed in parallel by multiple threads, will always be sequenced somehow, i.e. one thread executes all the method, then the next thread, …

```
synchronized public void addScore (int v) {
    score = score + v;
    scoreLabel.setText("Score = " + score);
}
```

Dynamic construction:

- More interesting if we can build up the pinball layout at runtime
- Rewrite MouseKeeper class:
  - Have a pallet of targets
  - If mousePressed() there, and mouseReleased() inside the target area, then put an appropriate target there
  - Communication uses data member element

Polymorphism

- A variable/parameter/data member declared some type can actually hold a subtype as well, e.g.:
  - public void hitBy(Ball aBall) { … }
  - but the actual Ball is really a PinBall (which extends Ball)
- Also interface can be used as type:
  - PinBallTarget target = (PinBallTarget) targets.elementAt(j);
  - but actually Wall, etc. (which implements PinBallTarget)

Summary

- Used collection class Vector
- MouseListener/MouseAdaptor
- Multiple Threads
- Exceptions
- Interface vs. inheritance
- Polymorphism: Ball / PinBall, …