

11. Testing Class *protoBug*

In the following class driver, not all *protoBug* methods are tested. The reader is invited to add the code necessary to fully test all *protoBug* methods. In this version of class *driver*, four *protoBug* objects are created and animated along fixed paths by a loop.

Because of the number of methods in *protoBug* and their complexity, testing *protoBug* requires a few extra techniques. These are described as follows.

In Java, the *break* statement works like it does in C++, exiting a control structure before code following *break* can be executed. In this version of *driver.java*, *break* will be used to halt a loop prematurely when two *protoBug* objects collide.

```
if (r.impactBug(s)) break;
```

(Note: Like C++, Java has a continue statement, which like the break statement causes following statements in a loop to be skipped. Unlike the break statement, the continue statement does not break out of the surrounding control structure. If that control structure is a loop, the loop will continue to repeat its body after having skipped the statements in the body that follow the continue statement.)

Two handy methods in Java very useful in testing and debugging are the *print* and *println* methods of the *out* class, which is part of the *System* package. *(Note: Packages act as grouping of classes, in some ways similar to projects in some C++ environments and in other ways like a C++ library.)* Method *println* outputs the results of expressions to the default output device (i.e., not a running applet), then changes lines. Method *print* does the same, but does not cause a line change. Using *print* and *println*, limited output can be directed to the command prompt window where it does not interfere with the layout of a running applet. The forms of *print* and *println* are as follows:

```
System.out.print(expression);
System.out.print(expression + expression);
System.out.print(expression + ... + expression);

System.out.println(expression);
System.out.println(expression + expression);
System.out.println(expression + ... + expression);
```

In this version of class *driver*, *print* will be used to output the values given by *protoBug* methods *returnx* and *returny* as follows.

```
System.out.print("p=(" + p.returnx() + "," + p.returny() + ")\t");
```

Notice that the character output commands found in C++ also work in Java.

Lastly, a method called *pause* is created and used in driver. Method *pause* is used to insert a delay between redrawing the objects displayed in the running applet. This is a common convention in animated applets that allows control of the speed of the animation refresh rate. Here, *pause* uses the Java control structure *try-catch* to make a call to the *sleep* method of class *Thread*. Method *sleep* requires one parameter, the length of time in milliseconds to pause the program. Method *sleep* may fail or encounter some difficulty at run time. This will generate an exception condition that must be handled by the program. The *try-catch* control structure is designed to allow the programmer to insert code to handle just such a failure. By calling *sleep* in a *try*, any exception generated will be intercepted and handled by the code in the *catch*. Notice that *catch* requires a parameter of class *Exception*.

```
private void pause (int time) {  
    try { Thread.sleep(time); }  
    catch (Exception e) {}  
}
```

In the case of the method *pause*, there is no need to perform any actions if an exception occurs. Thus *catch* does not contain any code. The example is using *try-catch* to protect the applet from the being damaged by an unhandled exception.

Complete Listing of the Current Version of *driver*

```

import java.awt.*;
import java.applet.Applet;

public class driver extends Applet {

    public void paint(Graphics g) {
        Background b = new Background(500,400);
        protoBug p = new protoBug(500,400,100,275);
        protoBug q = new protoBug(500,400,300,50);
        protoBug r = new protoBug(500,400,200,350);
        protoBug s = new protoBug(500,400,200,50);

        p.setBugColor(0,0,255);
        q.setBugColor(255,0,0);
        r.setBugColor(0,255,255);
        s.setBugColor(255,255,0);

        b.paint(g); p.paint(g); q.paint(g); r.paint(g); s.paint(g);

        for (int j=0; j<8; j++) s.turnleft( );

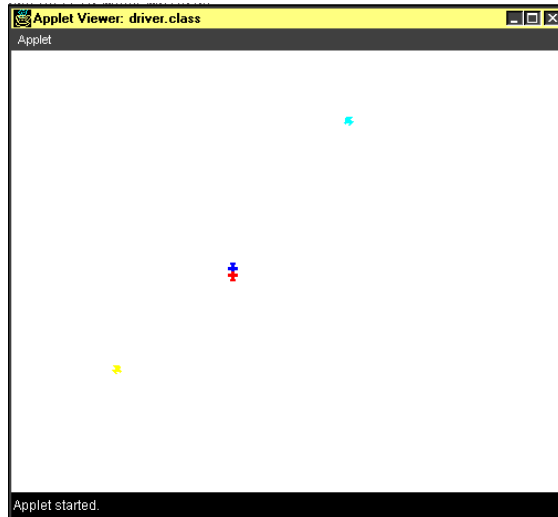
        for (int i=0; i<100; i++) {
            p.turnleft(); q.turnright();
            p.go( ); q.go( ); r.go( ); s.go( );
            b.paint(g); p.paint(g); q.paint(g); r.paint(g); s.paint(g);
            System.out.print("p=(" + p.returnx( ) + "," +
                               p.returny( )+")\t");
            if (r.impactBug(s)) break;
            pause(50);
        }
    }

    private void pause (int time) {
        try { Thread.sleep(time); }
        catch (Exception e) {}
    }
}

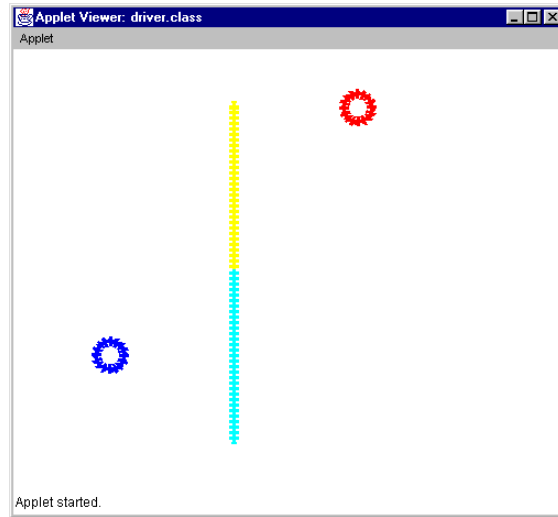
```

Compiling *protoBug.java* and *driver.java*, then running *driver.class* produced the following:

With Colors Reversed



Without a Background Object



```
C:\jdk1.3\projects\bugs>go
```

```
C:\jdk1.3\projects\bugs>del driver.class
File not found
```

```
C:\jdk1.3\projects\bugs>del protoBug.class
File not found
```

```
C:\jdk1.3\projects\bugs>javac protoBug.java
```

```
C:\jdk1.3\projects\bugs>pause
Press any key to continue . . .
```

```
C:\jdk1.3\projects\bugs>javac driver.java
```

```
C:\jdk1.3\projects\bugs>pause
Press any key to continue . . .
```

```
C:\jdk1.3\projects\bugs>appletviewer driver.html
```

```
p=(98,270)    p=(95,267)    p=(90,265)    p=(85,265)    p=(80,267)
p=(77,270)    p=(75,275)    p=(75,280)    p=(77,285)    p=(80,288)
p=(85,290)    p=(90,290)    p=(95,288)    p=(98,285)    p=(100,280)
p=(100,275)   p=(98,270)    p=(95,267)    p=(90,265)    p=(85,265)
p=(80,267)    p=(77,270)    p=(75,275)    p=(75,280)    p=(77,285)
p=(80,288)    p=(85,290)    p=(90,290)    p=(95,288)    p=(98,285)
```