Java Vocabulary

• Polymorphism continued - a subclass' version of a polymorphic method must be able to override the inherited method of its super-class.

• Concrete Class - a class encapsulating defined methods only, which can be instantiated as a new object. Java programs require the use of at least one concrete class.

• Abstract Class - a class encapsulating defined methods and/or mere declarations, which cannot be instantiated as a new object. Java programs do not require the use of abstract classes. Methods can be abstract as well.

• Interface - a reference, or entry point, into an abstract method. Standard interfaces include: Cloneable (exact duplication, not just double instantiation - Ex. Date rightNow = new Date(); Date rightNowClone = (Date) rightNow.clone();), Serializable (creation of a byte stream), and Runnable (default - execute at time of creation). – Ex. public abstract interface Runnable { public abstract void go(); }

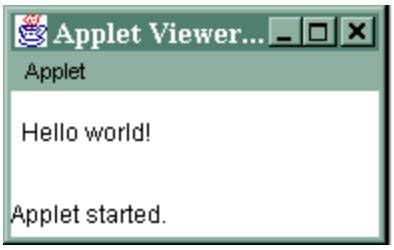
• Package - a library of classes that can be "imported" for compilation with the source code.

• Event Handling - the processing of an event -- e.g. a mouse click, movement of a scrollbar, or a key press -- captured by an event Listener. – Ex. Public void actionPerformed(ActionEvent e) { String cmd = e.getActionCommand();}

- Exception - an unexpected event or process that may or may not be fatal to a program's continued execution. Exceptions are usually exposed, and handled, by use of a try-catch block. – Ex. try { ServerSocket servSock = new ServerSocket(port);} catch(IOException e) { System.err.println(e);}

Applet Dissection

```
import java.awt.Graphics;
import java.applet.*;
public class MyHelloWorld extends Applet
{
        public void paint(Graphics g) { g.drawString("Howdy world!", 5, 25); }
}
```



Inheritance and Polymorphism Example

```
public class FlyingThing
{
        private String name;

        public String name( )
        {
                return name;
        }
        public void setName(String ft )
        {
                name = ft;
        }
        public String changeAltitude(int alt)
        {
                return = “The new altitude is “ + alt;
        }
}

public class Airplane extends FlyingThing
{
        public String changeAltitude(int alt)
        {
                return “  changed its altitude to “ + alt;
        }
}

public class Bird extends FlyingThing
{
        public String changeAltitude(int alt)
        {
```

```
            return "  flapped to an altitude of " + alt;
        }
}

public class FlyingThingApp
{
        public static void main( )
        {
            Airplane p = new Airplane( );
            p.setName("Citation");
            System.out.println(p.getName( ) + p.changeAltitude(165));
            // The output will be:
            // Citation changed its altitude to 165

            Bird b = new Bird( );
            b.setName("Polly");
            System.out.println(b.getName( ) + b.changeAltitude(12));
            // The output will be:
            // Polly flapped to an altitude of 12

            // Now consider
            FlyingThing ft = new Airplane( );
            ft.setName("Bob");
            System.out.println(ft.getName( ) + ft.changeAltitude(90));
            // The output will be:
            // Bob changed its altitude to 90

            ft = new Bird( );
            ft.setName("PollyAgain");
            System.out.println(ft.getName( ) + ft.changeAltitude(8));
            // The output will be:
            // PollyAgain flapped to an altitude of 8

            // Notice that the object ft was only declared once, yet
            // referenced two different subclasses of FlyingThing
            // This is polymorphism, and the methods called are
            // referred to as polymorphic methods.
        }
}
```