1.1's Object Enhancements

- Reflection
  - in java.lang.reflect package
  - ability of a class to "have a look around"
  - new classes:
    - Field
    - Method
    - Constructor
    - these are final classes—only the Java VM can allocate them

1.1's Object Enhancements

- class java.lang.Class expanded
  - now returns info about any type, not just classes
  - wrapper class constants
    - Integer.TYPE
      - Class object representing the primitive type int
    - Void.TYPE, Short.TYPE, Boolean.TYPE, etc.
  - new shorthand attributes: "class literals"
    - int.class ≡ Integer.TYPE
    - String.class ≡ Class.forName ("java.lang.String")
- extra facilities vital for JavaBeans, RMI and serialization

1.1's Object Enhancements

```
import java.lang.reflect.*;
import java.io.*;

class Reflect
  {
  static PrintWriter stdout = new PrintWriter (System.out);
  public static void main (final String [] args)
    {
    try
      {
      Class theClass = Class.forName (args [0]);
      Method [] methods = theClass.getMethods ();
      for (int i = 0; i < methods.length; i ++)
        stdout.println ("Method: " + methods [i].toString ());
      stdout.flush ();
      }
    catch (final ClassNotFoundException cnfe)
      {
      stdout.println (cnfe);
      System.exit (1);
      }
    }
  }
```

1.1's Object Enhancements

```
>java Reflect java.lang.System
Method: public static void java.lang.System.setIn(java.io.InputStream)
Method: public static void java.lang.System.setOut(java.io.PrintStream)
Method: public static void java.lang.System.setErr(java.io.PrintStream)
Method: public static void
  java.lang.System.setSecurityManager(java.lang.SecurityManager)
Method: public static java.lang.SecurityManager
  java.lang.System.getSecurityManager()
Method: public static native long java.lang.System.currentTimeMillis()
Method: public static native void
  java.lang.System.arraycopy(java.lang.Object,int,java.lang.Object,int,int)
Method: public static native int
java.lang.System.identityHashCode(java.lang.Object)
Method: public static java.util.Properties java.lang.System.getProperties()
Method: public static void
java.lang.System.setProperties(java.util.Properties)
Method: public static java.lang.String
  java.lang.System.getProperty(java.lang.String)
...
```

1.1's Object Enhancements

- Inner Classes
  - major addition
  - classes can be declared inside other classes and even locally within a block or expression
  - not actually supported by the VM—compiler 'trickery'
    - filenames: Outer$Inner.class
  - *"Have the language designers finally gone off the deep end?"*
  - four new types of class
    - nested
    - member
    - local
    - anonymous

1.1's Object Enhancements

- the purpose of these new inner class enhancements is to avoid "namespace pollution" by allowing a definition to be placed as close to the site of use as possible
- need experience to use correctly
  - not a 'traditional' feature
  - Smalltalk users should be slightly more comfortable, however!

1.1's Object Enhancements

- Nested top-level classes

```
public class Outer
  {
  …
  static class Inner
    {
    …
    }
  }
```

  - always static
  - can be named directly from the top level
    - Outer.Inner
  - Inner can access Outer's static state/members

1.1's Object Enhancements

- provides a convenient package-like structure, especially for multiple classes
- can be nested to any depth
- inner classes (of any kind) cannot declare any static members
- nested interfaces also possible
  - always implicitly static

1.1's Object Enhancements

- Member classes

```
class Test
  {
  int x = 0;
  class Inner1
    { private int y = x; }
  Inner1 in = new Inner1 ();
  int z = in.y;
  class Inner2
    { int y = z, a = in.y; }
  }
```

  - each instance of a member class associated with a containing instance
  - fields inside Inner1 accessible to Outer
    - even if private! *(buggy in JDK 1.1.x!)*
    - Inner2 can access Inner1's fields as well

1.1's Object Enhancements

- can't be named from the top level
  - equivalent to private fields
- can't have member interfaces
- some new syntax additions to allow class to identify its enclosing instance
  - Inner1.this.y
  - *containing_instance*.new
    - Fuselage fuselage = new Fuselage ();
    - Fuselage.Wing wing = fuselage.new Wing ();
    - Fuselage.Wing.Engine engine = wing.new RB299 ();
- class has a hidden private field pointing to enclosing instance

1.1's Object Enhancements

```
public class Employee
  {
  private double salary;
  private String name;

  private class WriteOnceSalaryProperty
    {
    private boolean isSet = false;

    public double get ()  { return (isSet ? salary : -1.0); }

    public void set (double salary)
      {
      if (isSet) return;
      if (salary > 0.0)
        {
        isSet = true;
        this.salary = salary ;
        }
      }
    }

  public WriteOnceSalaryProperty getSalaryProperty ()
    { return (new WriteOnceSalaryProperty ()); }
  }
```

1.1's Object Enhancements

```
public class Employee
  {
  private double salary;
  private String name;

  private class WriteOnceSalaryProperty
    {
    Employee outer;
    public WriteOnceSalaryProperty (Employee e)
      { outer = e; }
    private boolean isSet = false;
    public double get ()  { return (this.isSet ? outer.salary : -1.0); }
    public void set (double salary)
      {
      if (this.isSet) return;
      if (salary > 0.0)
        {
        this.isSet = true;
        outer.salary = salary ;
        }
      }
    }

  public WriteOnceSalaryProperty getSalaryProperty ()
    { return (new WriteOnceSalaryProperty ()); }
  }
```

behind the scenes code

1.1's Object Enhancements

```
D:\Bob\Test>javap -p Employee$WriteOnceSalaryProperty
Compiled from Employee.java
private synchronized class Employee$WriteOnceSalaryProperty
  extends java.lang.Object
    /* ACC_SUPER bit set */
{
    private final Employee this$0;
    private boolean isSet;
    public double get();
    public void set(double);
    Employee$WriteOnceSalaryProperty(Employee);
}
```

1.1's Object Enhancements

- Local classes
  - class definition is local to a block

```
Enumeration myEnumerate (final Object array [])
  {
  class E implements Enumeration
    {
    int count = 0;
    public boolean hasMoreElements ()
      { return count < array.length; }
    public Object nextElement ()
      { return array [count++]; }
    }
  return new E ();
  }
```

  - may access any names which are available to ordinary expressions within the same block
  - after function returns, array is still available until all references disappear

1.1's Object Enhancements

- note required use of final in parameter
  - mechanism can only work if accessible state (locals/parameters) can't ever change value
  - hence use of final keyword—to ensure state is a constant
  - state is copied into inner classes: *"…compiler automatically generates data fields for every accessed outer variable"*
- local classes can't use new syntax for 'this' and 'new'

1.1's Object Enhancements

- Anonymous classes
  - class definition is local to an *expression*

```
Enumeration myEnumerate (final Object array [])
  {
  return new Enumeration ()
    {
    int count = 0;
    public boolean hasMoreElements()
      { return count < array.length; }
    public Object nextElement()
      { return array[count++]; }
    }; // note: semicolon ends statement
  }
```

```
addActionListener
(
new ActionListener ()
  {
2 public void actionPerformed (ActionEvent evt)
    { System.exit (0); }
  }
);
```

  - an abbreviated notation for local classes
  - a single expression combines the definition of an anonymous class with the allocation of the instance

1.1's Object Enhancements

- can't have a constructor
  - since no name!
  - instance initializers instead

```
class X
  {
  int [] array = new int [10];
    { for (int i = 0; i < 10; i ++) array [i] = i; }
```

    - code blocks run at instantiation time
      - in source code order
- anonymous & local classes very valuable for AWT's listeners/adapters and other situations where delegation requires short, isolated code segments

1.1's Object Enhancements

```
import java.awt.*; import java.awt.event.*;

public class ObfuscatedHoHoHo
  {
  public static void main (String [] args)
    {
    new Frame ()
      {
      public void show ()
        {
        setSize (300, 300);
        add ("South", new Button ("Close")
          {
          public void addNotify ()
            {
            super.addNotify ();
            addActionListener
              (
              new ActionListener ()
                {
                public void actionPerformed (ActionEvent evt)
                  { System.exit (0); }
                }
              );
            }
          });
        super.show ();
        }
      }.show ();
    }
  }
```