Figure 10.0 Wrapper classes.

A consequence of rule 4 on the previous slide is that you can't immediately store a primitive **int** in a Vector—you first need to "wrap it up" in a class:

```
int i = 10;
Integer iWrap = new Integer(i);
```

Integer is a *wrapper class*. Here's how to envision **i** and **iWrap**:

Now that **iWrap** is an object, you can store it as usual …

```
Vector list = new Vector();
list.addElement(iWrap);
```

and retrieve it as usual …

```
________ getIback = ________ list.elementAt(0);
```

but to get the primitive **int** back out, you need to invoke the **intValue**() method of class **Integer** …

```
int j = getIback.intValue();
```

You can do all of this more briefly by combining statements:

```
int i = 10;
Vector list = new Vector();
list.addElement(new Integer(i));
int j = ((Integer)list.elementAt(0)).intValue();
```

but you need to be careful with parenthesis, because casts have lower precedence than method calls.

Exercise: What happens if you write the last statement as:

```
int j = (Integer)list.elementAt(0).intValue();
```