Figure 16.2 The *instanceof* operator.

Even though with polymorphism Java automatically selects the correct subclass for us, there are still times when we need to know that subclass; i.e. there are still times when we need to know the exact class an object belongs to. For example, let's print the student numbers of all **Student**s in a list of **Person**s.

```
public class Person { … }

public class Student extends Person {
    …
    public String getStudentNumber() {
        return stunum;
    }
    …
}
```

And in **main()** we have:

```
Person[] list = new Person[ …];
// instantiate and initialize the elements of list,
// some will be Students, others will be Persons

// now print student numbers
for (int i = 0; i < list.length; i++)
    System.out.println(list[i].getStudentNumber());     // this fails
```

The compiler rejects this because **list[i]** is a **Person** and class **Person** has no **getStudentNumber()** method.

With the **instanceof** operator, we can apply the method call only to **Student**s:

```
for (int i = 0; i < list.length; i++)
    if (list[i] instanceof Student)
        System.out.println(list[i].getStudentNumber());     // this still fails
```

We know **list[i]** is a **Student**, but the compiler knows it only as a **Person**. We need a *cast*:

```
for (int i = 0; i < list.length; i++)
    if (list[i] instanceof Student)
        System.out.println(((Student)list[i]).getStudentNumber());     // success!
```

Notice the parentheses around the cast. A cast has lower precedence than a method call.