Ad-hoc polymorphism

Ad-hoc\(^1\) polymorphism in Java occurs when a method or operator is applicable to different types. We look at three kinds of ad-hoc polymorphism: (1) overloading of methods, (2) overloading of operator +, and (3) autoboxing / unboxing.

Overloading names

Class Math in package java.lang has lots of functions for performing basic numeric operations. You will no doubt use class Math often. Four of its functions are shown to the right. They calculate the absolute value of a double value, a float value, an int value, and a long value.

All four functions are named abs! In certain circumstances, several methods can have the same name. This is called overloading.

This is an example of polymorphism: function abs calculates an absolute value, and it comes in four forms. Remember, polymorphism is the capability of assuming different forms.

It’s good that overloading is allowed in Java. If not, we would have to have four different names for the four abs methods. That could get messier as we write larger programs with many similar methods.

When there is a method call like Math.abs(-5) in a program, the argument, -5, is used to determine which function to call. Since -5 has type int, the function with an int parameter will be called.

You may wonder whether you can overload method names in your programs. Yes, of course! We illustrate this with class Counter to the right, which you might have written. An instance of Counter maintains a counter. The counter is initially 0, and it can be incremented with a call on procedure increment.

There are two reset procedures. One sets the counter to 0, the other sets the counter to its parameter i. The name reset is overloaded. Both procedures reset the counter, but they have different forms. That is polymorphism.

Class Counter illustrates another form of polymorphism and overloading. The name ctr takes two forms: a variable and a method. The field name is ctr, and method ctr() returns the value of field ctr. In Java, a variable and method can have the same name.

Note that overloading and overriding are different. Look up overriding in JavaHyperText.

Overloading operators

Another form of overloading is operator overloading. The operator + has several meanings: addition and catenation. For example,

The value of 2 + 3 is 5; \( \text{operator } + \text{ stands for int addition.} \)

The value of 2.0 + 3 is 5.0; \( \text{operator } + \text{ stands for double addition since 2.0 is a double, the 3 an int.} \)

The value of "2" + 3 is "23"; since at least one operand is a String, + stands for string catenation.

The programming language Python allows the programmer to overload operators like + and – and * with other meanings. Java, however, does not allow this.

---

\(^1\) “ad hoc”: for the particular end or case at hand without consideration of wider application [Merriam-Webster]
Ad-hoc Polymorphism

Autoboxing/unboxing

If you are unfamiliar with the notion of a wrapper class, before reading further, read the one-page pdf file under entry wrapper in JavaHyperText.

Primitive type values are not objects, so int value 2110 is not an object. However, there are places in a Java program where one would like to have an int value but only an object can appear. Therefore, Java has wrapper class Integer, each object of which wraps an int value. Thus, as shown below, an integer comes in two forms: as a primitive value and as a primitive value wrapped in an Integer object. That’s polymorphism.

If an integer like 2110 appears in a place where an object is required, Java will automatically wrap it in an Integer object. It will also automatically unwrap it.

Java calls these two coercions autoboxing and auto-unboxing instead of auto-wrapping and auto-unwrapping. We give examples: autoboxing will happen in the first assignment to the right and auto-unboxing in the second one.

You can write your own code to wrap and unwrap primitive values. As an example, the call shown to the right returns a pointer to an Integer object that wraps the integer 2110.2 In the second example shown to the right, first, 2110 is wrapped and the pointer to the object is stored in k; then the following method call returns the int value that is in object k.

Summary

We have discussed three kinds of ad-hoc polymorphism:

- Method name overloading: a method has “many forms”. For each call, the types of the arguments are used to distinguish which form of the method to call.
- Operator overloading: + can be int addition, long addition, double addition, float addition, or String catenation. The types of the operands in an expression op1 + op2 determine which it is.
- Autoboxing: A primitive value has two forms: the value itself and the value wrapped in an object. Java automatically wraps and unwraps it for you where the need arises.

Note that this kind of polymorphism is a compile-time feature, a syntactic feature, not a runtime feature. For example,

- the compiler will look at a call of an overloaded method and determine which form of the method to call.
- The compiler will determine what kind of an operator an occurrence of + is, based on the types of its operands.
- The compiler will insert autoboxing or auto-unboxing operations where necessary.

The ad-hoc polymorphism features greatly simplify programming.

---

2 There is a constructor for class Integer, but it has been deprecated — it is preferable not to use it. The Java API documentation for the wrapper classes say to use instead of the constructor the static function valueOf (e.g. Integer.valueOf (3), Boolean.valueOf (true)) because valueOf is likely to yield significantly better time and space performance than the constructor.