Class definition / declaration

```
public class C {
    private int b;

    public int getB() {
        return b;
    }
}
```

A class (or class definition, or class declaration) is a blueprint that describes the contents of each object of the class. To the right is a definition of class *C*. Use it as a first model for any class that you write. We explain its pieces.

Keyword **public** is an *access modifier*. It indicates that all parts of a program can access class *C*. The first brace { and last brace } delimit the *body* of the class. The body contains two declarations:

- A declaration of variable b. It is called a *field* of the class. Access modifier **private** indicates that this field can be accessed or referenced only from within objects of this class, not from outside the class.
- A declaration of function *getB*. It is **public**, so it can be referenced or called from outside the class. It returns an **int** value. Its body, delimited by { and }, contains the statement **return** *b*;.

The order of the declarations in a class doesn't matter at all, but, the convention is to place the field declarations first, then constructors, then getter methods, then setter methods, and then others. This is a guideline, not a requirement.

Above, we said that the class is a blueprint that describes the contents of each object of the class. To the right we draw an object of the class, based on this blueprint.

It looks like a manila folder. The tab at the top contains the name of, or a pointer to, the object itself. The tab contains (1) the name of the class, (2) @, and (3) an integer written in hexadecimal. When *we* draw an object, we put any integer we want after @ to distinguish it from other objects. When a computer creates the object during program execution, it puts the address in memory of the object, written in hexadecimal.
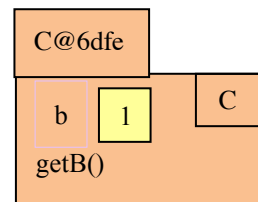
We put a small box with the class name in the upper right.

The class definition contains a declaration of field *b*. Therefore, variable b is in the object; here, it happens to contain the value 1.

The class definition contains a declaration of method *getB*. Therefore, method *getB* is in the object. We write only the method signature "*getB*()", but actually the whole method is there, and instead of *getB*() we might write:

> *getB*() {…}   or   *getB*() { **return** b; }

Thus, every object of class *C* contains all the fields and methods that are declared in class *C*.

Note: Generally, a public class *C* that you write is placed in a file *C.java* and stored on your hard drive.

Subclasses and superclasses

```
public class S extends C {
    private int f = 2;

    public void setF(int x) {
        f = x;
    }
}
```

To the right is a declaration of another class *S*. It is different from the declaration of *C* in that it has an *extends clause*,
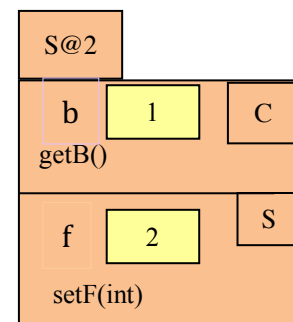
> **extends** C

Thus, an object of class *S* contains not only declared te fields and methods but also *all the fields and methods of class C*. Here is some terminology:

- *S* is a *subclass* of *C*.
- *C* is a *superclass* of *S*.
- Subclass *S inherits* all fields and methods of superclass *C*.

We draw an object of class *S* to the right. Now there are two *partitions*, a partition for the components (fields and methods) declared in class *C* and a partition for the components declared in class *S*. The partition for *S* appears under the partition for *C* since *S* is the subclass and *C* is the superclass.

The declaration of a subclass is a great way to use previously written code. With just the introduction of "**extends** *C*", we get to use all that *C* has to offer. This is a major feature of object-oriented programming.

Class Object, the superest class of them all

Java has a predeclared class *Object*, in package *java.lang*. Any class that does not explicitly extend another class automatically extends class *Object*. Since class *C* declared on the previous page does not explicitly extend another class, it automatically extends Object.

Therefore, the objects of class *C* and *S* on the previous page should be drawn as shown below. Note that object *S@2* has *three* partitions: the top one for superclass *Object*, the middle one for its subclass *C*, and the lower one for *C*'s subclass *S*.

However, to reduce clutter, when there is no reason to draw attention to class *Object*, we may omit its partition.

Object has no superclass above it, so we call it the *superest* class of them all.

Class Object declares about 11 methods. The objects below show the two you will learn about first: *equals*() and *toString*(). They are discussed elsewhere.