Smalltalk has a very simple model of inheritance

Here is an instance method.

```
| anObject |
anObject := B new.
anObject m1
```

local variable
class name
class method
receiver
message

```
Class A superType Object
    instance variables : x
    instance methods :
        getX
            ^x

Class B superType A
    instance variables : y
    instance methods :
        getY
            ^y
```

every instance variable is private
every instance method is public

Smalltalk implements inheritance using a run-time search.

1. The receiver attempts to match the message with one of it's class's methods.
2. If a match is found, its body is run and an object is returned
3. If a match is not found, the message search continues in the superclass (and any other superclasses up the inheritance chain)
4. If the method is not found in Object, then a system error occurs.

Method polymorphism in C++ (and Java)

- Late binding, where late ≡ run-time
- Smalltalk allows polymorphism because any message can be sent to any object

```
class A {
public:
   void f1();
};

class B : public A {
public
   void f1();
};

class C : public A {
public:
   void f1();
};
```

```
A a1;
B b1;
C c1;

a1.f1();
b1.f1();
c1.f1();
```

} strong typing ensures that the f1 in the class of the receiver is called

```
a1 = b1;
a1.f1();
```

← still calls f1 in A

```
a1 = c1;
a1.f1();
```

← calls f1 in A

What about pointers?

```
A *pa1 = new A();
B *pb1 = new B();
C *pc1 = new C();
```

```
pa1->f1();
pb1->f1();
pc1->f1();
```

} call f1 in class corresponding to the object

```
pa1 = pb1;
pa1->f1(); <- still call f1 in A
```

If we want to call f1 in B through pa1, then we need to declare f1 in A as "virtual"

```
class A {
public:
    virtual void f1();
};
```

```
A *pa1 = new A();
B *pb1 = new B();

pa1 -> f1();   <- no call compiles but code to look up the correct
                  function at run-time — calls f1 in A

pa1 = pb1;
pa1 -> f1();   <- calls f1 in B
```

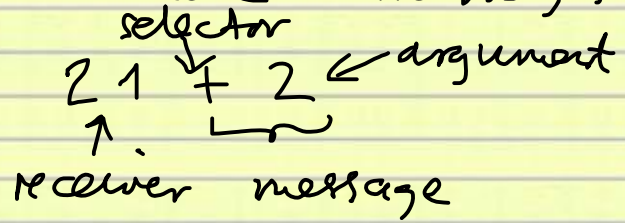
The compiler deposits RTTI so that the type of every object is stored with the object. Also a table of pointers to functions where virtual functions are declared. (The VTable)

This call (to f1) would fail if the corresponding function is not available

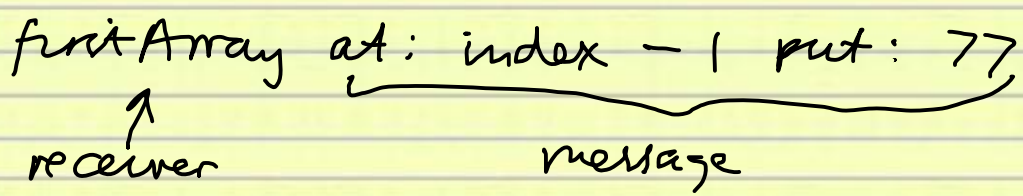
We can make a virtual function "pure" by giving it a null body : `virtual void f1() = 0;`

This makes the class abstract (no objects can be made)

Smalltalk : messages and methods

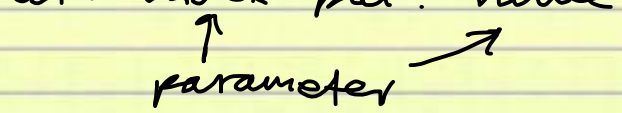


We can also have keyword messages



The message selector is in two parts at: put:

The method might be declared as:



It's exactly like firstArray[index-1] = 77;

* In general a message alters the instance variables of the receiver — the receiving object changes state.

Methods are stored in classes:

general form:

pattern [| local vars |] statement_body

e.g.

currentTotal — unary message

^ (oldTotal + newValue).

x: xCoord y: yCoord

aturtle up.
aturtle goto: xCoord @ yCoord.
aturtle down.

sending the x:y: message

obj x: 300 y: 400

Statements can be put into blocks, which are objects – they can be assigned to variables.

```
[ index := index + 1. sum := sum + index ]

summer := [ ... ]

summer value
```

↑ runs code in the block

Iteration in Smalltalk

```
count := 1.
sum := 0.
[ count <= 20 ]
    whileTrue: [ sum := sum + count.
                 count := count + 1 ]
```