Object-Oriented Programming

- Class is a form of abstract type
- Instances of class called **objects**
- Operators of class called **methods**
- Applying a **method** to an object in the class called **message passing**
  - Messages take object as implicit argument

Variables and Methods

- Classes have two kinds of methods and two kinds of variables:
  - Instance methods and instance variables
  - Class methods and class variable

Instance Variables and Methods

- Instance variables:
  - Instance variables hold state of single object
  - Different objects have different (copies of) instance variables
  - sometimes called *fields*
- Instance methods:
  - Instance methods act on a single object, usually acting on instance variables

Class Variables and Methods

- Class variables hold state that is shared in common among all objects of class
- Class methods act on attributes of the whole class through class variables
- Class methods can’t access instance variables
- Will only use instance variables and methods here

Creating Objects

- Class usually has an *initialization* method (usually called the same as the class, or **initialize**)
- Creates an object of class **c** by

  new c*(args)*
- Creates new instance variables; executes code in def of initialization method

Sending a message

- A message is a method is a function
- Sending a message to an object is invoking that method on that object is applying the function to the object
- Common notation: **o . m**(*args*)
- Apply **m** to **o** and additional *args*

Self

- Methods have formal parameters for all arguments accept ojbect to which it is sent
- Problem: How to access that argument?
- Answer: use **self**

Example

```
class a
  { var i ;
     var j ;
     method a( )
     { i := #1 ; j := i + #1 ; return j };
     method f( ) { return self . a( ) };
     method g( ) { return self . f( ) };
     method h( ) { return (i + j) }
}
```

Self and Recursion

```
class oddeven
{ method oddeven( ) { return true }
  method odd(n)
  {if n = 0 then return false
    else self . even (n - 1) };
  method even(n)
   {if n = 0 then return true
    else self . odd (n - 1) }
}
```

Subclasses

- Classes form a hierarchy of parent classes and children classes
- Child is a subclass of, derived class of, inherits from, extends the parent, superclass
- Common syntax:

class c extends c'

Inheritance

- **Inheritance** allows all variables and methods exported from **parent** class to be part of a **subclass**

Inheritance

- Export information:
  - **public**: everybody sees
  - **private**: only seen inside class
  - **protected**: exported only to subclasses
- We will use only public for methods protected for variables

Single Versus Multiple Inheritance

- Single inheritance: Class may only be a subclass of one parent
- Multiple inheritance: Class may be immediate subclass of two or more parents
- Problem with multiple inheritance:

  class A:B,C {…}

  What if we have both B.m and C.m? Which method is A.m?

Answer: Language dependent, usually B.m (first extended)

Variable Hiding and Static Scope

- Instance variables of superclass may be *hidden* by declarations in subclass

```
class a { var i; var j; …}
class b extends a
{var j; var k; method f(){…} …}
class c extends b { … }
```

- Instance variables available to methods of b are i from a and j, k from b
- Will not change when method of b sent to object of c

Method Hiding and Super

- Instance method of subclass may hide (or shadow) method of superclass
- Access hidden method **m** of parent class by using **super m (***args***)**

Example

```
class b extends a
{ var j ; var k ;
  method a() { return self . b() }
  method b()
  { call super a() ; j = #10 ; k = j + #1 ;
     return k }
  method g() { return super h() }
  method h() { return self . g() } }
```

Example

```
class c extends b
 { method a() { return super a() }
    method b() { return super b() }
    method c() { i = #100 ; j = i + #1 ; k =
  j + #1 ; return k } method g() { return
  (i + k * j) } }
```

Inheritance Polymorphism

- If a method is added to a subclass with the same name (and usually signature) as in the parent class the new version **overrides** inherited version in subclass

Inheritance Polymorphism and Dynamic Dispatch

- Method of superclass may be overridden in subclass
- **Dynamic dispatch:** when a method of superclass is applied to object of subclass, methods of subclass are used in computing result, instead of methods of superclass

```
class A { var i; method int:h (); method int: f ( x) {x + h();};
          method A () {i = 3} }
class B extends A { method g() {f(i)}   method h() {4;}  … }
```

Virtual (or Abstract) Methods

- **Virtual method:** has declaration but no function body

- For subclass to be instantiated it must override all virtual methods with methods with fully concrete bodies