

Programación orientada a objetos: Simula y Smalltalk

Paradigmas de la Programación

FaMAF – UNC 2021

capítulo 11

basado en filmas de Vitaly Shmatikov

Simula 67

- primer lenguaje orientado a objetos
- una extensión de Algol 60 para simulación, pero luego se reconoce como de propósito general
- estandarizado en 1977
- inspiración para muchos otros, en particular, smalltalk y C++

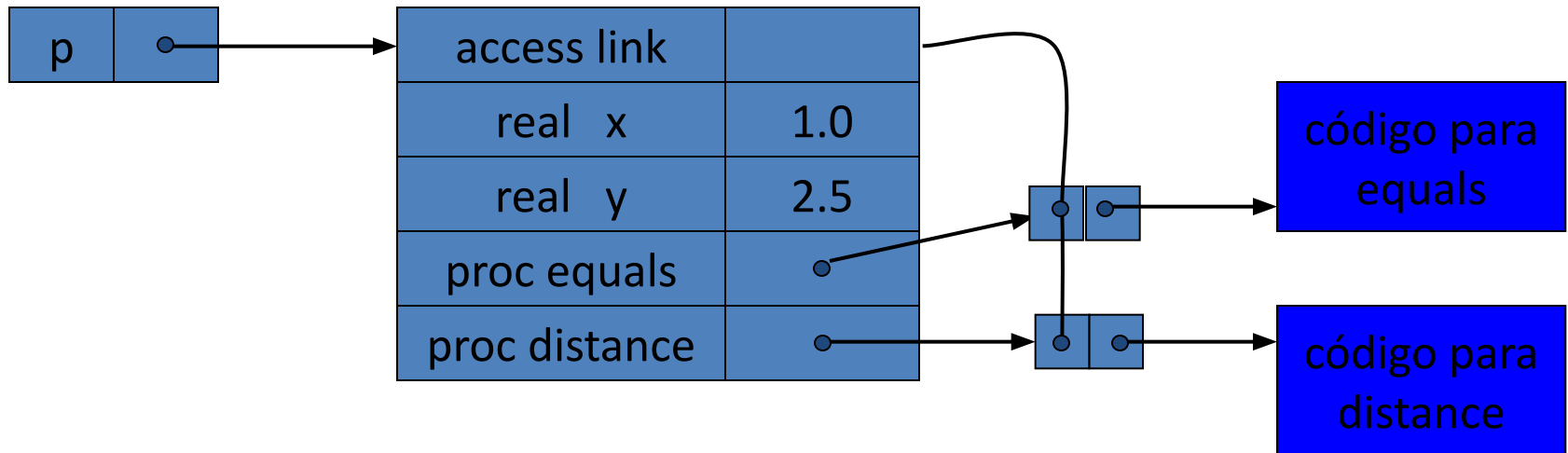
historia

- lenguaje de simulación
- centro noruego
- Dahl, Myhrhaug, Nygaard
 - Nygaard era un especialista en investigación operativa y activista político, pretendía que
 - los lenguajes pudieran describir sistemas industriales y sociales
 - la gente común pudiera entender cambios políticos

objetos en Simula

- **clase**
 - un procedimiento que devuelve un puntero al activation record en el que se ejecuta
- **objeto**
 - activation record que se genera al llamar a una clase
- **acceder un objeto**
 - acceder cualquier variable o procedimiento local
- **manejo de memoria**
 - recolección de basura

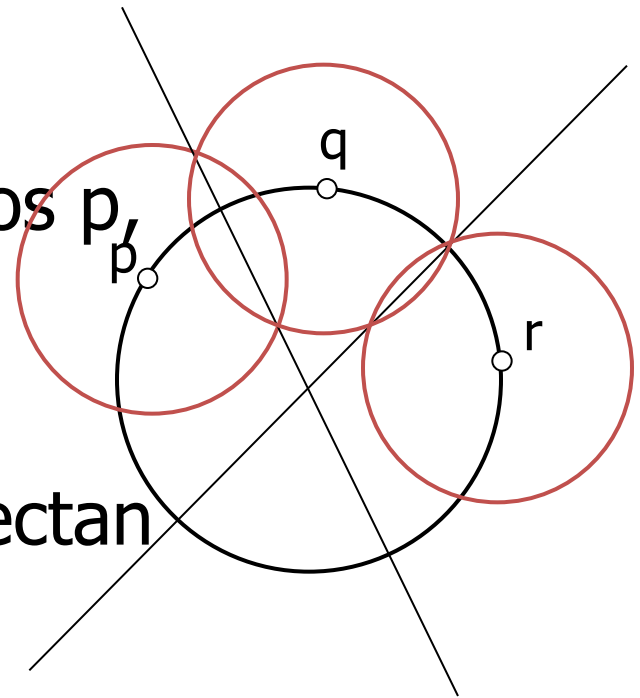
cómo se representan los objetos



Un objeto se representa con un activation record con un access link para encontrar las variables globales con alcance estático

ejemplo: círculos y líneas

- problema
 - encontrar centro y radio del círculo que pasa por los puntos p , q y r
- solución
 - dibujar círculos que se intersectan
 - dibujar líneas a través de la intersección de los círculos
 - la intersección de las líneas es el centro del círculo que se busca



modelar esto en simula

- los puntos, líneas y círculos son objetos
- operaciones asociadas a los objetos
 - punto
 - equality(anotherPoint) : boolean
 - distance(anotherPoint) : real
 - línea
 - parallelto(anotherLine) : boolean
 - meets(anotherLine) : REF(Point)
 - círculo
 - intersects(anotherCircle) : REF(Line)

clase punto en simula

```
class Point(x,y); real x,y;
```

```
begin
```

```
  boolean procedure equals(p); ref(Point) p;
```

```
    if p /= none then
```

```
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
```

```
  real procedure distance(p); ref(Point) p;
```

```
    if p == none then error else
```

```
      distance := sqrt(( x - p.x )**2 + (y - p.y) ** 2);
```

```
end ***Point***
```

```
p := new Point(1.0, 2.5);
```

```
q := new Point(2.0,3.5);
```

```
if p.distance(q) > 2 then ...
```

el argumento **p** es un puntero a **Point**

un **ptr** no inicializado
tiene el valor **none**

asignación de puntero

clase línea en simula

```
class Line(a,b,c); real a,b,c;  
begin  
  boolean procedure parallelto(l); ref(Line) l;  
    if l /= none then parallelto := ...  
  ref(Point) procedure meets(l); ref(Line) l;  
    begin real t;  
      if l /= none and ~parallelto(l) then ...  
    end;  
  real d; d := sqrt(a**2 + b**2);  
  if d = 0.0 then error else  
    begin  
      d := 1/d;  
      a := a*d; b := b*d; c := c*d;  
    end;  
end *** Line***
```

← variables locales

una línea definida
por $ax+by+c=0$

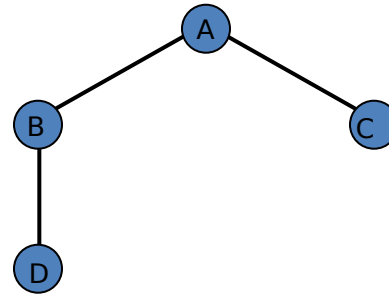
procedimientos

inicialización

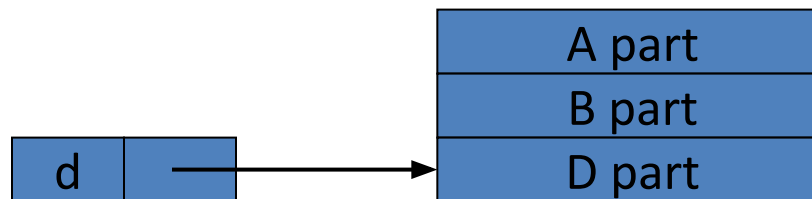
clases derivadas en simula

- cuando se declara una clase se le puede prefijar el nombre de otra clase

```
class A  
A class B  
A class C  
B class D
```



- un objeto de una clase “prefijada” es la concatenación de objetos de cada clase del prefijo



subtipado

- el tipo de un objeto es su clase
- el tipo de una subclase se trata como un subtipo del tipo asociado con la superclase
- ejemplo:
 - class A(...); ...
 - A class B(...); ...
 - ref (A) a :- new A(...)
 - ref (B) b :- new B(...)
 - a := b /* legal porque B es una subclase de A */
 - b := a /* también legal, pero hay que comprobarlo en tiempo de ejecución*/

principales características orientadas a objetos

- clases
- objetos
- herencia (“prefijado de clases”)
- subtipado
- métodos virtuales: se puede redefinir una función en una subclase

qué NO tenía Simula 67

- encapsulación: se pueden acceder todos los datos y funciones
- sin mecanismo self/super (a diferencia de Smalltalk)
 - pero se puede usar la expresión `this <class>` para referirse al objeto en sí mismo
- sin variables de clase, pero con variables globales
- sin excepciones

resumen de Simula

- una clase es un procedimiento que devuelve un puntero a un activation record, el código de inicialización se ejecuta siempre como cuerpo del procedimiento
- un objeto es una clausura creada por una clase
- sin encapsulación
- subtipado mediante jerarquía de clases
- herencia por prefijado de clases

Programación orientada a objetos: Smalltalk

Paradigmas de la Programación

FaMAF – UNC 2020

capítulo 11

basado en filmas de Vitaly Shmatikov

Smalltalk

- el lenguaje importante que popularizó objetos
- desarrollado en Xerox PARC
- extiende y desarrolla la metáfora de objetos
 - algunas ideas de Simula, pero muy distinto
 - **todo es un objeto**, incluso una clase (como en Lisp “todo es una lista”)
 - todas las operaciones son mensajes a objetos
 - muy flexible y poderoso: si un objeto que recibe un mensaje que no entiende, trata de inferir qué puede hacer

la aplicación que motivó el desarrollo: Dynabook

- una computadora chica, portable
- para esa aplicación, Smalltalk debía ser
 - un lenguaje de programación e interfaz al sistema operativo
 - orientado a “no programadores”

Smalltalk hoy

- <http://www.fast.org.ar/>



terminología Smalltalk

- **objeto** instancia de una **clase**
- **clase** define el **comportamiento** de sus **objetos**
- **subclase** **clase** definida como **modificaciones incrementales** a una superclase
- **selector** **nombre** de un **mensaje**
- **mensaje** **selector** con **valores para sus parámetros**
- **método** **código** que usa una **clase** para **responder a un mensaje**
- **variable de instancia** datos guardados en un objeto

tipos de objetos

- cada objeto tiene una interfaz
 - interfaz = métodos de instancia declarados en la clase

ejemplo:

```
Point      { x:y:, move Dx:Dy:, x, y, draw}  
ColorPoint { x:y:, move Dx:Dy:, x, y, color, draw}
```

- es una forma de tipo
 - sólo los nombres de los métodos, nada sobre los argumentos
- uso de objetos con tipo
 - cuando se envía un mensaje a un objeto...

```
p draw          p  x:3  y:4  
q color         q  move Dx: 5 Dy: 2
```

la expresión anda si el mensaje está en la interfaz

encapsulación en Smalltalk

- los métodos son públicos
- las variables de instancia están ocultas
 - invisibles para otros objetos...
 - ... pero las pueden manipular los métodos de subclase
 - esto limita la forma de establecer invariantes
 - ejemplo:
 - una superclase mantiene una lista ordenada de mensajes con algún selector, por ejemplo, insert
 - una subclase puede acceder esta lista directamente y reordenarla

ejemplo: clase punto

nombre	Point
superclase	Object
variable de clase	pi
variable instancia	x y
mensajes y métodos de la clase	
〈...nombres y código de los métodos...〉	
mensajes y métodos de la instancia	
〈...nombres y código de los métodos...〉	

mensajes y métodos de la clase

```
new X:xvalue Y:yvalue | |  
    ^ self new x: xvalue y: yvalue  
  
newOrigin | |  
    ^ self new x: 0 y: 0  
  
initialize | |  
    pi <- 3.14159
```

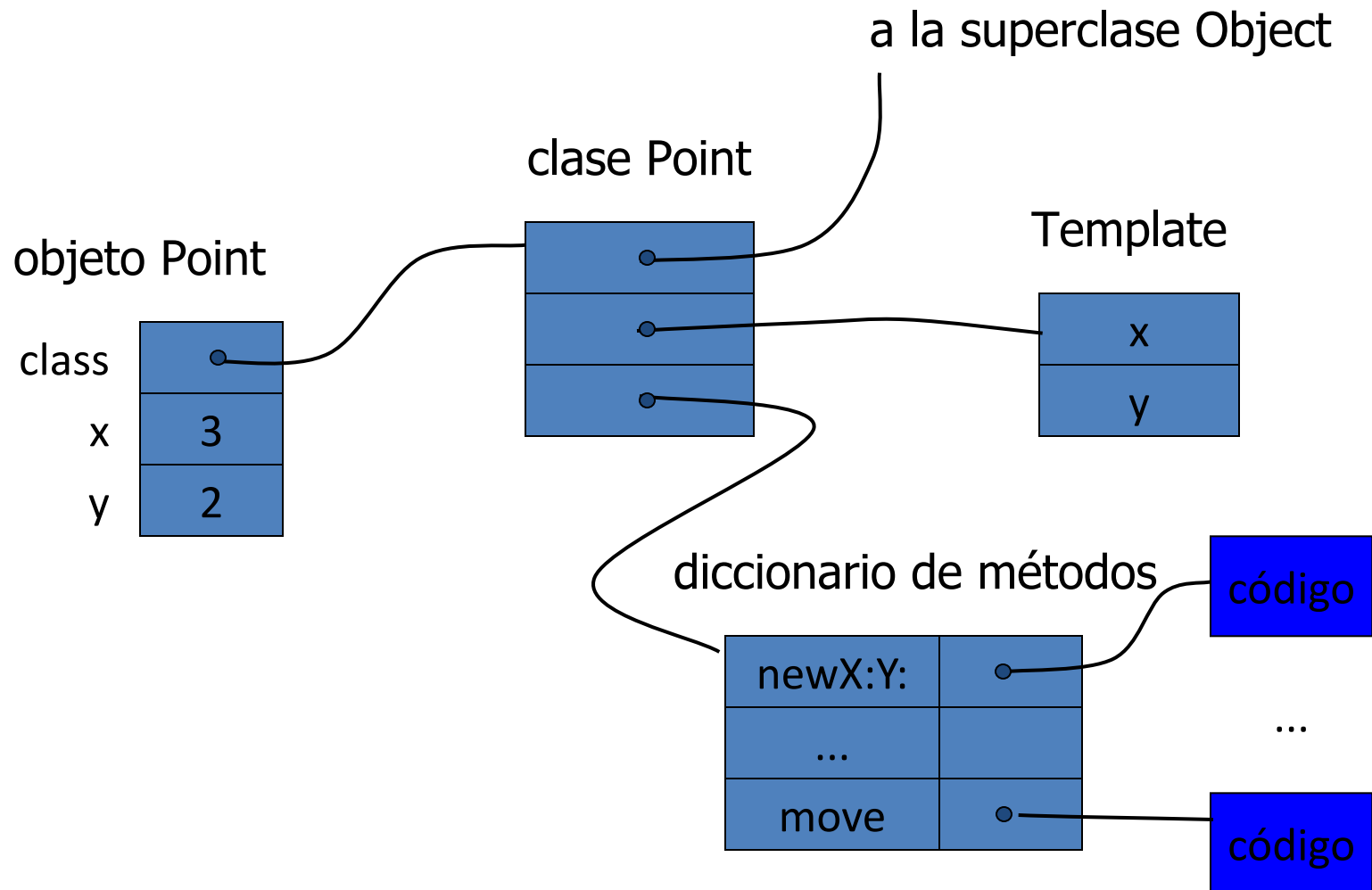
- el selector es `newX:Y:`, por ejemplo `Point new X:3 Y:2`
- `^` marca el valor de retorno
- `| |` marca el alcance de una declaración local
- `<-` es asignación
- `new` es un método para toda clase, heredado de `Object`
- el método `initialize` fija `pi`

mensajes y métodos de la instancia

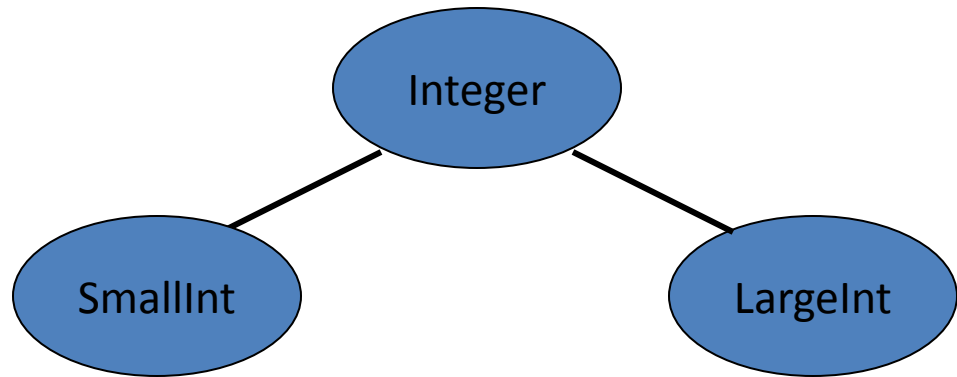
```
x: xcoord y: ycoord | |  
    x <- xcoord  
    y <- ycoord  
move Dx: dx Dy: dy | |  
    x <- dx + x  
    y <- dy + y  
x | | ^x  
y | | ^y  
draw | |  
    <...código para  
dibujar point...>
```

- se instancian las coordenadas x e y, e.g., pt x:5 y:3
- se mueve point en la cantidad establecida
- se devuelve la variable de instancia oculta x
- se devuelve la variable de instancia oculta y
- se dibuja el punto en la pantalla

representación de Point en tiempo de ejecución



Self y Super



```
Factorial | |  
    self <= 1  
        ifTrue: [^1]  
        ifFalse: [^(self-1) factorial * self ]
```

herencia y subtipado

herencia

definir puntos coloreados a partir de puntos

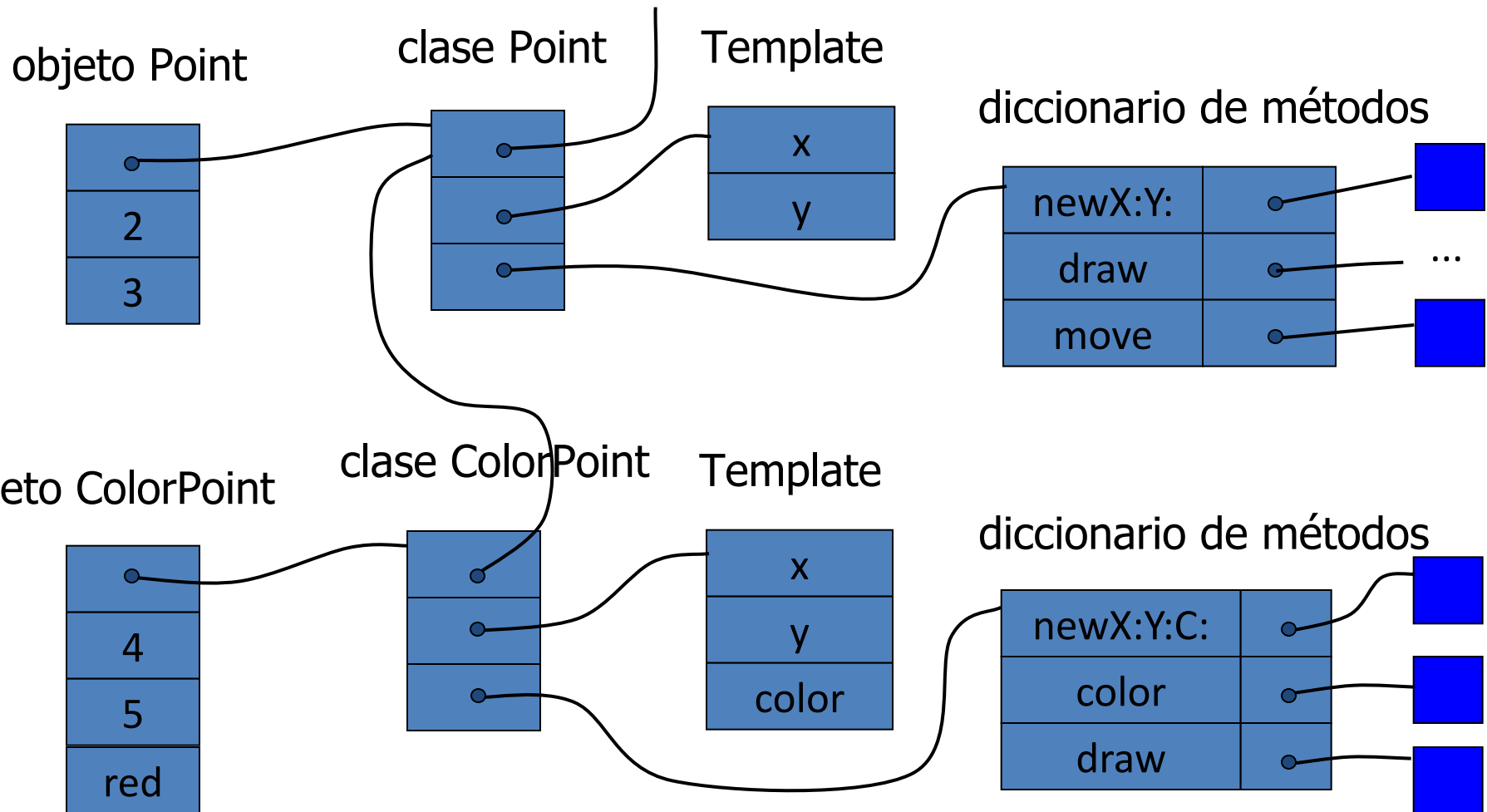
nombre de clase	ColorPoint
superclase	Point
variables de clase	
variables de instancia	color
métodos y mensajes de clase	
newX:xv Y:yv C:cv	<... código...>
métodos y mensajes de instancia	
color	^color
draw	<... código...>

nueva variable
de instancia

nuevo método

sobreescribe el
método de Point

representación en tiempo de ejecución



esto es un esquema conceptual, las implementaciones pueden ser muy distintas

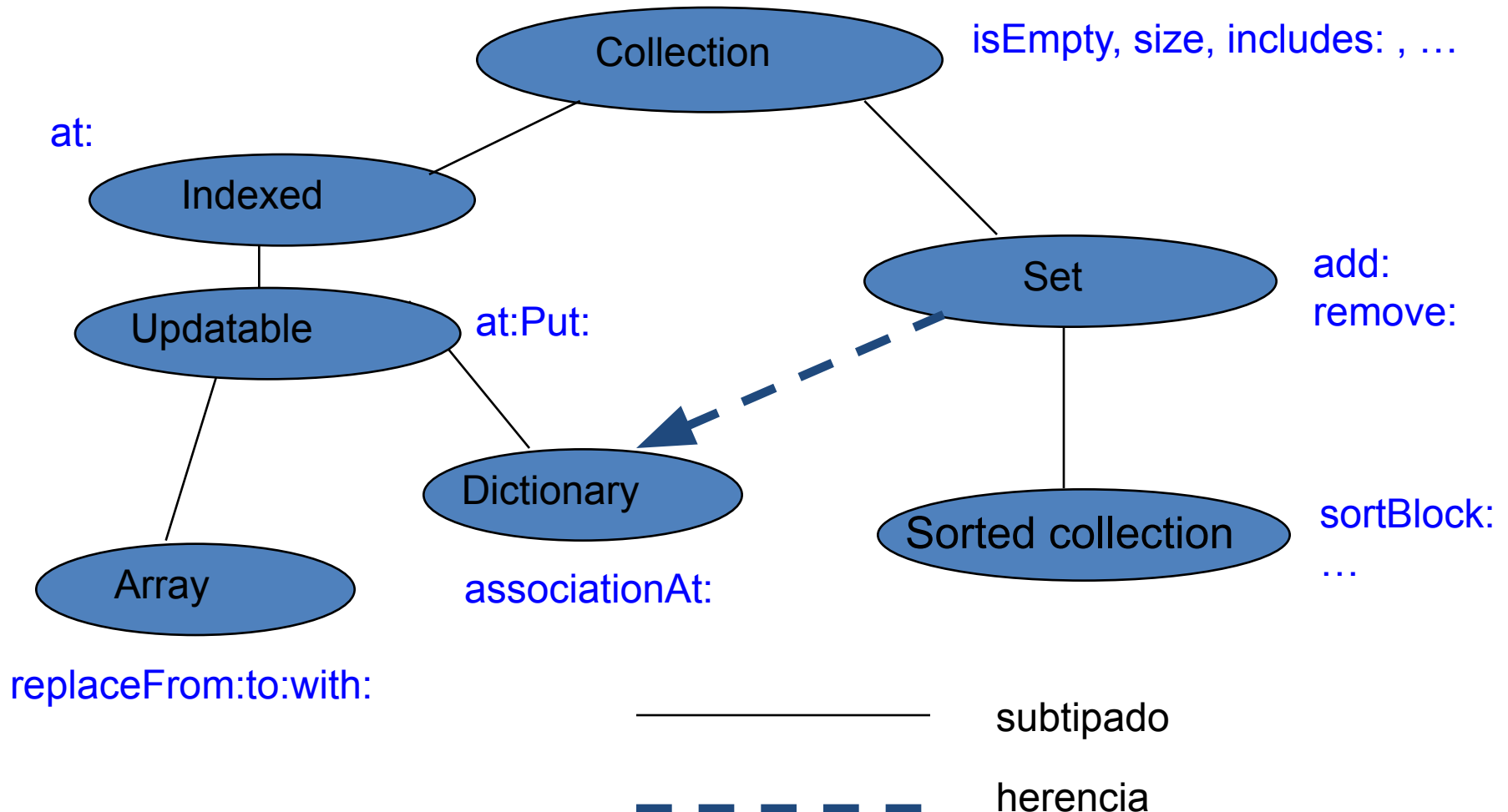
subtipado

- relación entre interfaces
 - supongamos que la expresión tiene sentido
p msg:params -- funciona si msg está en la interfaz de p
 - sustituimos p por q si la interfaz de q contiene a la interfaz de p
- subtipado
 - si la interfaz es un superconjunto, entonces es un subtipo
 - ej: ColorPoint es un subtipo de Point
 - a veces llamado “conformidad”

Subtipado y herencia

- el subtipado es implícito
 - no es parte del lenguaje
 - es un aspecto importante de cómo se construyen los sistemas
- la herencia es explícita
 - se usa para implementar sistemas
 - no fuerza la relación a subtipado

jerarquía de collection



flexibilidad y expresividad

flexibilidad de Smalltalk

- expresividad: se pueden definir las construcciones del lenguaje en el lenguaje mismo?
 - Lisp cond: Lisp permite formas especiales definidas por el usuario
 - ML datatype: suficiente para definir listas polimórficas, equivalentes al tipo lista built-in
 - ML overloading: no está disponible para el programador
- Smalltalk es expresivo en este sentido
 - muchas construcciones primitivas en otros lenguajes se pueden definir en Smalltalk (e.g., Booleanos y Bloques)

Booleanos y Bloques Smalltalk

- el valor Booleano es un objeto con `ifTrue:` `ifFalse:`
 - clase `boolean` con subclases `True` y `False`
 - `True ifTrue:B1 ifFalse:B2` ejecuta `B1`
 - `False ifTrue:B1 ifFalse:B2` ejecuta `B2`

expresión de ejemplo:

```
i < j ifTrue: [i add 1] ifFalse: [j subtract 1]
```

- `i < j` es expresión booleana, produce un objeto booleano
- los argumentos son bloques, objetos que ejecutan métodos
- los booleanos y los bloques son muy comunes
 - hay una optimización para booleanos
 - sintaxis especial para bloques

test de Ingalls

- Dan Ingalls: diseñador principal del sistema Smalltalk
 - recibió el premio Grace Murray Hopper por su trabajo en Smalltalk y gráficos Bitmap en Xerox PARC
- propone un test para saber si algo es “orientado a objetos”
 - se puede definir un nuevo tipo de entero, poner tus nuevos enteros en rectángulos (que ya son parte del sistema de ventanas), pedirle al sistema que oscurezca un rectángulo, y que todo funcione?
 - Smalltalk pasa, C++ falla

operaciones de enteros en Smalltalk

- expresión de enteros

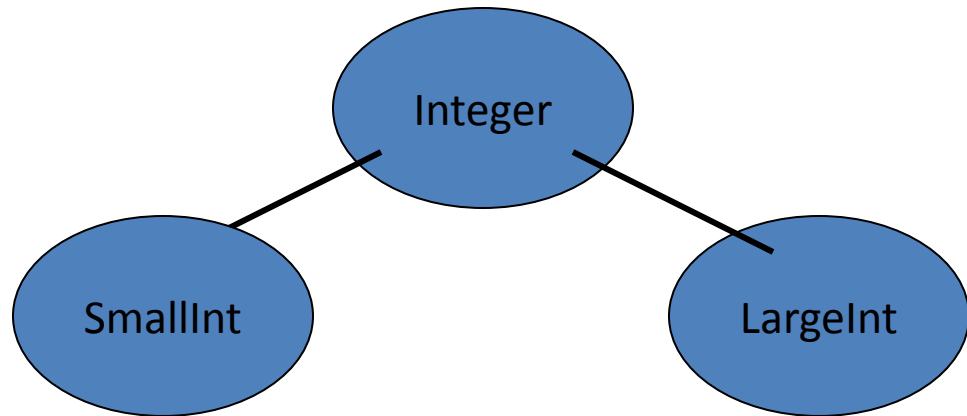
```
x plus: 1 times: 3 plus: (y plus: 1) print
```

- propiedades

- todas las operaciones se ejecutan enviando mensajes
- si `x` es de algún nuevo tipo de entero, la expresión tiene sentido siempre que `x` tenga los métodos `plus`, `times`, `print`

en realidad el compilador tiene algunas optimizaciones hardcodeadas, pero se revierte a esto si `x` no es un entero built-in

Self y Super



```
Factorial | |  
    self <= 1  
        ifTrue: [^1]  
        ifFalse: [^(self-1) factorial * self ]
```

- este método se puede implementar en Integer, y funciona incluso si SmallInt y LargeInt se representan distinto
- los sistemas de tipos de C++ y Java no toleran esto

costes y beneficios del “verdadero OO”

- por qué sirve el test de Ingalls?
 - asegura que todo es un objeto
 - se acceden los objetos sólo desde la interfaz
 - facilita la extensión de los programas
- cuál es el coste de implementación?
 - cada operación sobre enteros requiere una llamada a métodos (a no ser que haya optimizaciones de compilador)
 - vale la pena?

resumen de Smalltalk

- clase: crea objetos que comparten métodos
 - punteros al template, diccionario, clase madre
- objetos: creados por una clase, contienen variables de instancia
- Encapsulación
 - los métodos son públicos, las variables de instancia son ocultas
- subtipado: implícito, sin sistema de tipos estático
- herencia: subclases, self, super