

Programación Orientada a Objetos en Java

IN1015 – Programación II: Aplicaciones computacionales
Ingeniería Civil Industrial – Universidad de Aysén
Prof. Enrique Urrea – enrique.urrea@uaysen.cl



Universidad
de Aysén

```
if(top!=self)
function calcWidth() {
  var wW = 0;
  if (typeof window.innerWidth == 'number') {
    wW = window.innerWidth;
  } else if (document.documentElement && documentElement.clientWidth) {
    wW = document.documentElement.clientWidth;
  } else if (document.body && document.body.clientWidth) {
    wW = document.body.clientWidth;
  }
  if (sH = document.documentElement.scrollHeight)
  var wH = window.innerHeight || documentElement.clientHeight ||
  wW = !document.all && (sH > wH) ? sH : wW;
  return wW;
}
```

Códigos para esta parte del curso

<http://gitlab.com/eurra/IN1015/>

Paquete **src/poo/**



Este cuadro mostrará el
ejemplo respectivo

Conceptos básicos

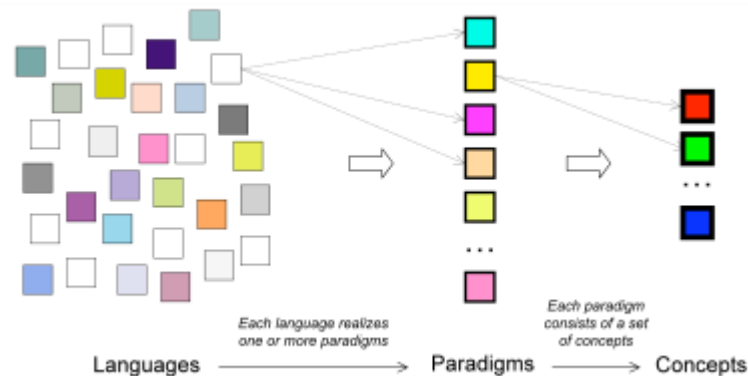
Introducción al paradigma



Universidad
de Aysén

¿Qué es un paradigma de programación?

- Conjunto de **conceptos** que permiten **resolver** de forma adecuada determinados **problemas** a través de la programación
- Estos conceptos se asocian a diversas **funcionalidades o herramientas** que pueden encontrarse en distintos lenguajes de programación
- Un paradigma define parte del “**modelo mental**” con el cual se trabaja en un lenguaje de programación.



Filas: Lenguajes

Columnas: Paradigmas

Language	Number of Paradigms	Concurrent	Constraints	Dataflow	Declarative	Distributed	Functional	Metaprogramming	Generic	Imperative	Logic	Reflection	Object-oriented	Pipelines	Visual	Rule-based	Other paradigms
LabVIEW	2	No	No	Yes	No	No	No	No	No	No	No	No	No	No	Yes	No	No
APL	2	No	No	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	No
ALF	2	No	No	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
DrRacket ^[citation needed]	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Sather ^[citation needed]	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Claire	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Spreadsheet	2	No	No	No	No	No	Yes	No	No	No	No	No	No	No	Yes	No	No
Amalg ^[citation needed]	2	No	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Simula ^[citation needed]	2	No	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Metaobject protocols	2	No	No	No	No	No	No	No	No	No	No	No	Yes ^{[a.1][a.2]}	No	No	No	No
Lava	2	No	No	No	No	No	No	No	No	No	No	No	Yes ^[a.1]	No	Yes	No	No
PointDragon	3	No	No	No	No	No	No	No	No	Yes	No	No	Yes	No	Yes	No	No
SISAL	3	Yes	No	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No
Erlang	3	Yes	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
ChucK ^[citation needed]	3	Yes	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Prograph	3	No	No	Yes	No	No	No	No	No	No	No	No	Yes ^[a.1]	No	Yes	No	No
Prolog	3	No	No	No	No	No	Yes	No	No	Yes	Yes	No	No	No	No	No	No
NET ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
J ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Perl ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Plan ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Tcl with Tcl or XOTcl extension ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
PHP ^{[2][3][4]}	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
ECMAScript ^{[5][6]} (ActionScript, E4X, JavaScript, JScript)	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Lua ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Tcl with Snit extension ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
C++	6 [14]	Yes ^{[2][8][9]}	Library ^[10]	Library ^{[11][12]}	Library ^{[13][14]}	Library ^{[15][16]}	Yes	Yes ^[17]	Yes ^[a.3]	Yes	Library ^{[18][19]}	Library ^[20]	Yes ^[a.1]	C++14 ^[21]	No	Library ^[22]	No
D (version 1.0)	3	No	No	No	No	No	No	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
D (version 2.0)	7	Yes	Partial	No	No	No	Yes	Yes	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
Embarcadero Delphi	3	Yes	No	No	No	No	No	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
E	3	Yes	No	No	No	Yes	No	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Gurx	4	Yes	Yes	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
Java	5	Yes	No	No	No	No	No	No	Yes	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Python ^[citation needed]	4	No	No	No	No	No	No	No	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	procedural
Ruby	4	No	No	No	No	No	Yes	No	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
OCaml	4	No	No	No	No	No	Yes	No	Yes	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Leda	4	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes ^[a.1]	No	No	No	No
ROOP	4	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	Yes	No	No
lg	4	Yes ^[a.4]	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
REBOL	4	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Fortran	5	Yes	No	No	No	No	Yes ^[a.5]	No	Yes ^[a.6]	No	No	No	Yes ^[a.1]	No	No	No	No
Ada ^{[23][24][25][26][27]}	5	Yes ^[a.7]	No	No	No	Yes	No	No	Yes	Yes	No	No	Yes ^[a.1]	No	No	No	No
Windows PowerShell	5	No	No	No	No	No	Yes	No	Yes	Yes	No	Yes	Yes ^[a.1]	Yes	No	No	No
Perl	5	No	No	No	No	No	Yes	No	Yes ^[a.3]	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Common Lisp (some other paradigms are implemented as libraries) ^[citation needed]	5	No	No	No	No	No	Yes	Yes	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Falcon	5	No	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes ^[a.1]	No	No	No	No
D (version 2.0) ^{[28][29]}	5	Yes ^[a.4]	No	No	No	No	Yes	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
Object Pascal	5	Yes	No	No	No	No	No	Yes	No	No	Yes	No	Yes ^[a.1]	No	No	No	No
Scala ^{[30][31]}	7	Yes ^[a.4]	No	Yes ^[a.8]	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Nemerle	7	Yes	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Ice	7	Yes	No	Yes ^[a.9]	No	No	No	Yes ^[a.10]	No	Yes	Yes	No	Yes ^[a.1]	No	No	No	No

Orientados
a Objetos

Java

Python

Language	Number of Paradigms	Concurrent	Constraints	Dataflow	Declarative	Distributed	Functional	Metaprogramming	Generic	Imperative	Logic	Reflection	Object-oriented	Pipelines	Visual	Rule-based	Other paradigms
LabVIEW	2	No	No	Yes	No	No	No	No	No	No	No	No	No	No	Yes	No	No
APL	2	No	No	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	No
ALF	2	No	No	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
DrRacket ^[citation needed]	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Sather ^[citation needed]	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Claire	2	No	No	No	No	No	Yes	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Soradashets	2	No	No	No	No	No	Yes	No	No	No	No	No	No	No	Yes	No	No
Ampart ^[citation needed]	2	No	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Simula ^[citation needed]	2	No	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Metaobject protocols	2	No	No	No	No	No	No	No	No	No	No	No	Yes ^{[a.1][a.2]}	No	No	No	No
Lava	2	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	No	No
PointDragon	3	No	No	No	No	No	No	No	No	Yes	No	No	Yes	No	Yes	No	No
SISAL	3	Yes	No	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No	No
Erlang	3	Yes	No	No	No	Yes	Yes	No	No	No	No	No	No	No	No	No	No
Chuck ^[citation needed]	3	Yes	No	No	No	No	No	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Prograph	3	No	No	Yes	No	No	No	No	No	No	No	No	Yes ^[a.1]	No	Yes	No	No
Prolog	3	No	No	No	No	No	Yes	No	No	Yes	Yes	No	No	No	No	No	No
BETA ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
J ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Perl ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Plan ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
Tcl with Tcl or XOTcl extensions ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
PHP ^{[2][3][4]}	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.1]	No	No	No	No
ECMAScript ^{[5][6]} (ActionScript, E4X, JavaScript, JScript)	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Lua ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Tcl with Snit extension ^[citation needed]	3	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
C++	6 [14]	Yes ^{[2][8][9]}	Library ^[10]	Library ^{[11][12]}	Library ^{[13][14]}	Library ^{[15][16]}	Yes	Yes ^[17]	Yes ^[a.3]	Yes	Library ^{[18][19]}	Library ^[20]	Yes ^[a.1]	C++ ^[21]	No	Library ^[22]	No
D (version 1.0)	3	No	No	No	No	No	No	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
D (version 2.0)	7	Yes	Partial	No	No	No	Yes	Yes	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
Embarcadero Delphi	3	No	No	No	No	No	No	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
E	3	Yes	No	No	No	Yes	No	No	No	No	No	No	Yes ^[a.1]	No	No	No	No
Curry	4	Yes	Yes	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
Java	5	Yes	No	No	No	No	No	No	Yes	No	Yes	No	Yes ^[a.1]	No	No	No	No
Python ^[citation needed]	4	No	No	No	No	No	No	No	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	procedural
Ruby	4	No	No	No	No	No	Yes	No	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
OCaml	4	No	No	No	No	No	Yes	No	Yes	Yes	No	No	Yes ^[a.1]	No	No	No	No
Leda	4	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes ^[a.1]	No	No	No	No
ROOP	4	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	Yes	No	No
lg	4	Yes ^[a.4]	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
REBOL	4	No	No	No	No	No	Yes	No	No	Yes	No	No	Yes ^[a.2]	No	No	No	No
Fortran	5	Yes	No	No	No	No	Yes ^[a.5]	No	Yes ^[a.6]	No	No	No	Yes ^[a.1]	No	No	No	No
Ada ^{[23][24][25][26][27]}	5	Yes ^[a.7]	No	No	No	Yes	No	No	Yes	Yes	No	No	Yes ^[a.1]	No	No	No	No
Windows PowerShell	5	No	No	No	No	No	Yes	No	Yes	Yes	No	Yes	Yes ^[a.1]	Yes	No	No	No
Perl	5	No	No	No	No	No	Yes	No	Yes ^[a.3]	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Common Lisp (some other paradigms are implemented as libraries) ^[citation needed]	5	No	No	No	No	No	Yes	Yes	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Falcon	5	No	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes ^[a.1]	No	No	No	No
D (version 2.0) ^{[28][29]}	5	Yes ^[a.4]	No	No	No	No	Yes	No	Yes ^[a.3]	Yes	No	No	Yes ^[a.1]	No	No	No	No
Object Pascal	5	Yes	No	No	No	No	No	Yes	No	No	Yes	No	Yes ^[a.1]	No	No	No	No
Scala ^{[30][31]}	7	Yes ^[a.4]	No	Yes ^[a.8]	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Yes ^[a.1]	No	No	No	No
Nemerle	7	Yes	No	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes ^[a.1]	No	No	No	No
Re	7	Yes	No	No	No	No	No	Yes ^[a.10]	No	Yes	No	Yes	Yes ^[a.1]	No	No	No	No



Conceptos básicos

¿Y la Programación Orientada a Objetos? (POO)

- ☐ Uno de los paradigmas más populares y utilizados en los lenguajes modernos de programación
- ☐ A grandes rasgos, se trata de
 - ☐ Definir **clases** que describen y representan conceptos del dominio que se quiere resolver, a través de sus **atributos** y de las **operaciones** que se pueden realizar sobre estos conceptos
 - ☐ Desde las clases definidas se crean y utilizan **objetos**, los cuales son las instancias específicas de las clases



Conceptos básicos

Programación Orientada a Objetos (POO)

Clase



Instanciar



Objetos



Conceptos básicos

Programación Orientada a Objetos (POO)

Clase BolaHelado



- **Color**
- **Sabor**



Objetos

- **Azul**
- **Chirimoya**



- **Naranja**
- **Lúcuma**



- **Rosado**
- **Frutilla**



Conceptos básicos

Una **aplicación en Java** que usa POO, por lo general, tiene al menos **dos partes**:

La definición de la(s) clase(s) asociada(s) al problema a resolver

BolaHelado.java

```
public class BolaHelado {  
    private String color;  
    private String sabor;  
  
    public BolaHelado(String c,  
                        String s) {  
        color = c;  
        sabor = s;  
    }  
  
    public String comer() {  
        return "Delicioso helado  
        de " + sabor + "!";  
    }  
}
```



AppBolaHelado.java

```
public class AplicacionBolaHelado {  
    public static void main(String[] args) {  
        BolaHelado bola1 ...  
        ...  
    }  
}
```

La aplicación donde se utiliza la clase (en donde se instancian objetos de ella)



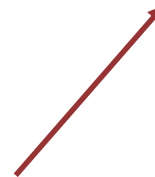
Conceptos básicos

Definición de una clase

```
public class BolaHelado {  
    private String color;  
    private String sabor;  
  
    public BolaHelado(String c,  
                        String s) {  
  
        color = c;  
        sabor = s;  
    }  
  
    public String comer() {  
        return "Delicioso helado  
            de " + sabor + "!";  
    }  
}
```



BolaHelado.java



Nota: Toda clase se define en su propio archivo fuente, **diferente** al de la aplicación en la cual se usa la clase



Conceptos básicos

Definición de una clase

```
public class BolaHelado {  
    private String color;  
    private String sabor;  
  
    public BolaHelado(String c,  
                        String s) {  
        color = c;  
        sabor = s;  
    }  
  
    public String comer() {  
        return "Delicioso helado  
            de " + sabor + "!";  
    }  
}
```

Atributos

- ☐ Representan las “**propiedades**” (datos) de la clase
- ☐ Se definen como **variables** + un modificador de **visibilidad** (ej.: **private**)



Conceptos básicos

Definición de una clase

```
public class BolaHelado {  
    private String color;  
    private String sabor;  
  
    public BolaHelado(String c,  
                        String s) {  
        color = c;  
        sabor = s;  
    }  
  
    public String comer() {  
        return "Delicioso helado  
            de " + sabor + "!";  
    }  
}
```

Constructores

- ☐ Operación que permite **instanciar** un objeto desde la clase
- ☐ Recibe los **parámetros** necesarios para **inicializar los atributos** de la clase



Conceptos básicos

Definición de una clase

```
public class BolaHelado {  
    private String color;  
    private String sabor;  
  
    public BolaHelado(String c,  
                        String s) {  
        color = c;  
        sabor = s;  
    }  
  
    public String comer() {  
        return "Delicioso helado  
               de " + sabor + "!";  
    }  
}
```

Métodos

- ☐ Operaciones que pueden ser invocadas sobre un objeto de la clase
- ☐ Por lo general, operan sobre los mismos atributos, ya sea utilizando sus valores o cambiándolos



Conceptos básicos

Utilización de una clase

```
public class AppBolaHelado {  
    public static void main(String[] args) {  
        BolaHelado bola1 = new BolaHelado("azul", "Chirimoya");  
        BolaHelado bola2 = new BolaHelado("Naranja", "Lúcuma");  
        BolaHelado bola3 = new BolaHelado("Rosado", "Frutilla");  
  
        System.out.println(bola1.comer());  
        System.out.println(bola2.comer());  
        System.out.println(bola3.comer());  
    }  
}
```



AppBolaHelado.java



Nota: Toda aplicación que utilice clases debe ir en el mismo package, o si no requerirá utilizar **import** para acceder a la clase



Conceptos básicos

Utilización de una clase

```
public class AppBolaHelado {  
    public static void main(String[] args) {  
        BolaHelado bola1 = new BolaHelado("azul","Chirimoya");  
        BolaHelado bola2 = new BolaHelado("Naranja","Lúcuma");  
        BolaHelado bola3 = new BolaHelado("Rosado","Frutilla");  
  
        System.out.println(bola1.comer());  
        System.out.println(bola2.comer());  
        System.out.println(bola3.comer());  
    }  
}
```



bola1

bola2

bola3



Creación de objetos

- La palabra **new** crea una **nueva instancia** de una clase, invocando el **constructor** con los **parámetros** que el mismo defina
- Para trabajar con los objetos, al crearlos, es necesario **asignarlos** a una **variable de referencia**



Conceptos básicos

Utilización de una clase

```
public class AppBolaHelado {  
    public static void main(String[] args) {  
        BolaHelado bola1 = new BolaHelado("azul", "Chirimoya");  
        BolaHelado bola2 = new BolaHelado("Naranja", "Lúcuma");  
        BolaHelado bola3 = new BolaHelado("Rosado", "Frutilla");  
  
        System.out.println(bola1.comer());  
        System.out.println(bola2.comer());  
        System.out.println(bola3.comer());  
    }  
}
```

`bola1.comer()`



Invocación de métodos

- ☐ Con las instancias creadas, es posible **invocar** los métodos declarados en la clase, usando el **operador de invocación** (punto)
- ☐ El **resultado** de la invocación dependerá del **objeto** utilizado en la invocación

¿Qué sale
por pantalla?



Manejo de estado

Actividad: Crear una clase `Alumno`, que tenga distintos atributos que Ud. crea convenientes, además de un método `public String presentarse()`, el cual retorne un texto con la presentación del alumno.

Luego, genere la aplicación `PrimerDiaClases`, en donde tenga al menos 3 referencias a alumnos, y en donde ellos se tengan que presentar.



Manejo de estado

Entendiendo las referencias



Universidad
de Aysén



Manejo de estado

Problema: Implemente una clase en Java que permita manejar la cuenta corriente un cliente en un banco. Cada cuenta tiene como atributos el RUT del cliente (String), el saldo actual y el número de transacciones (giros y/o depósitos) realizados sobre la cuenta.

Al crear una cuenta, se debe solicitar el RUT respectivo. El saldo y el número de transacciones parten en 0.

Implemente métodos en la clase para:

- ☐ Obtener el saldo y el número de transacciones de la cuenta
- ☐ Depositar una cantidad de dinero en la cuenta
- ☐ Girar una cantidad de dinero en la cuenta, sólo si hay saldo suficiente.



CuentaCorriente.java



Manejo de estado

Desarrollo: atributos y constructor

```
public class CuentaCorriente {  
    private String rut;  
    private int saldo;  
    private int numTransacciones;
```

Único dato necesario
para crear una cuenta

```
    public CuentaCorriente(String rut) {  
        this.rut = rut;  
        saldo = 0;  
        numTransacciones = 0;  
    }
```

Estas líneas son opcionales
(por defecto, un `int` es
inicializado en 0 en un objeto)


```
// ...
```



Manejo de estado

Desarrollo: método para obtener saldo

```
public int getSaldo() {  
    return saldo;  
}
```



Cómo estándar, un método que sólo retorna el valor de un atributo se denomina “getter”, y su nombre se construye como **get + nombre atributo**

La sentencia **return** permite retornar un resultado a quién invoque el método. El tipo del dato retornado debe coincidir con el tipo de retorno especificado en el método (**int**)



Manejo de estado

Desarrollo: método para girar

```
public boolean girar(int monto) {  
    if(saldo - monto >= 0) {  
        saldo -= monto;  
        numTransacciones++;  
        return true;  
    }  
  
    return false;  
}
```

Cómo se observa, es posible incorporar **validaciones** en los métodos, de forma que, acorde al estado de sus atributos, las operaciones se ejecuten de una u otra forma

Note que la sentencia **return** termina inmediatamente con la ejecución del método completo



Manejo de estado

Ahora... ¿Cómo **usamos** las cuentas?

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...





Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador

¿Qué imprime por pantalla?

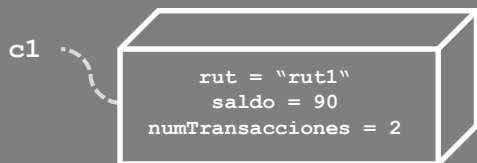
```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo());
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

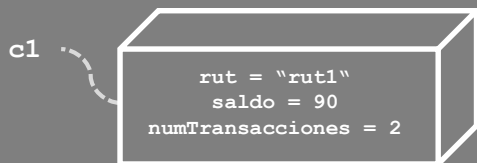
```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo()); // 90
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo());
```

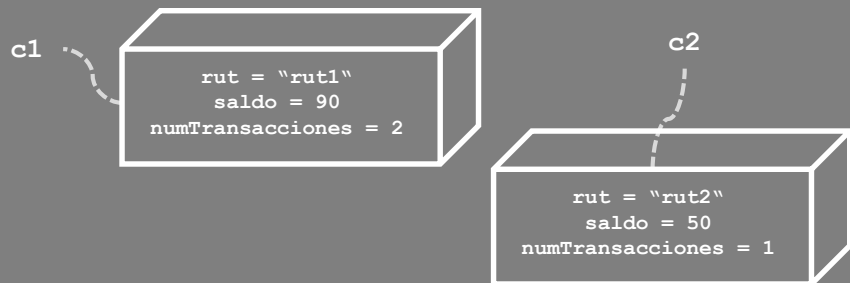
```
CuentaCorriente c2 = new CuentaCorriente("rut2");  
c2.depositar(50);  
c2.girar(70);  
System.out.println(c2.getSaldo());
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo()); // 90
```

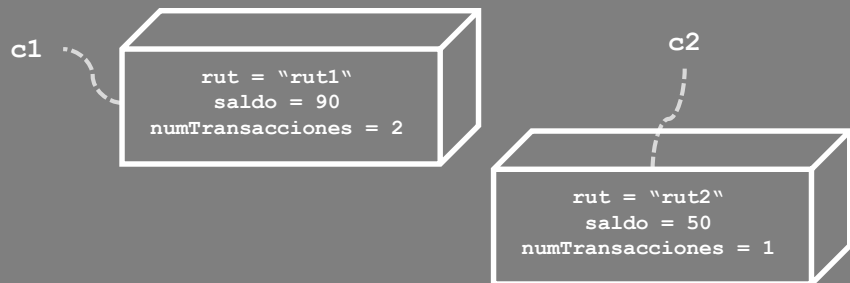
```
CuentaCorriente c2 = new CuentaCorriente("rut2");  
c2.depositar(50);  
c2.girar(70);  
System.out.println(c2.getSaldo()); // 50
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");
c1.depositar(100);
c1.girar(10);
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");
c2.depositar(50);
c2.girar(70);
System.out.println(c2.getSaldo()); // 50
```

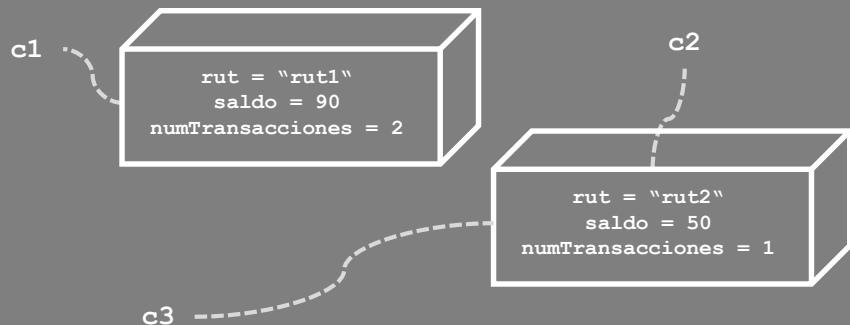
```
CuentaCorriente c3 = c2;
System.out.println(c3.getSaldo());
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");
c1.depositar(100);
c1.girar(10);
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");
c2.depositar(50);
c2.girar(70);
System.out.println(c2.getSaldo()); // 50
```

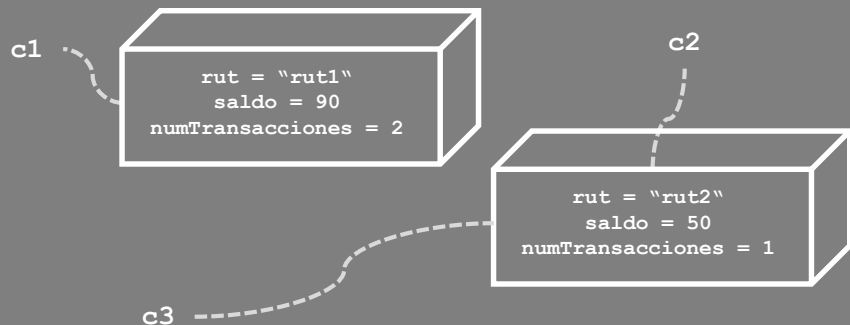
```
CuentaCorriente c3 = c2;
System.out.println(c3.getSaldo()); // 50
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");  
c2.depositar(50);  
c2.girar(70);  
System.out.println(c2.getSaldo()); // 50
```

```
CuentaCorriente c3 = c2;  
System.out.println(c3.getSaldo()); // 50
```

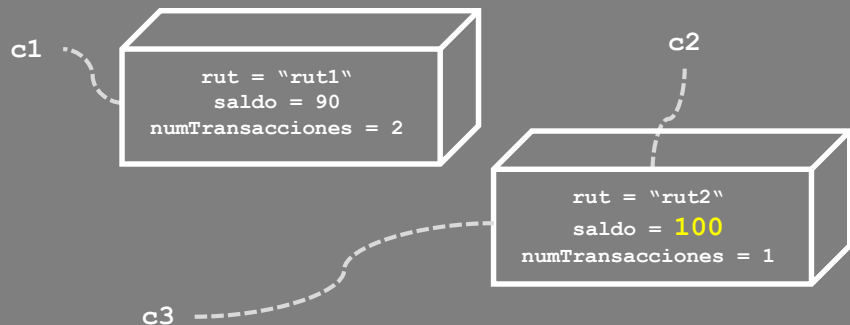
```
c3.depositar(50);  
System.out.println(c2.getSaldo());
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");  
c1.depositar(100);  
c1.girar(10);  
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");  
c2.depositar(50);  
c2.girar(70);  
System.out.println(c2.getSaldo()); // 50
```

```
CuentaCorriente c3 = c2;  
System.out.println(c3.getSaldo()); // 50
```

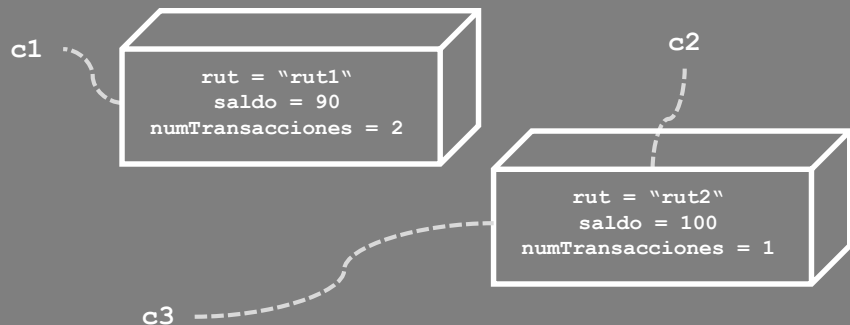
```
c3.depositar(50);  
System.out.println(c2.getSaldo()); // 100
```




Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");
c1.depositar(100);
c1.girar(10);
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");
c2.depositar(50);
c2.girar(70);
System.out.println(c2.getSaldo()); // 50
```

```
CuentaCorriente c3 = c2;
System.out.println(c3.getSaldo()); // 50
```

```
c3.depositar(50);
System.out.println(c2.getSaldo()); // 100
```

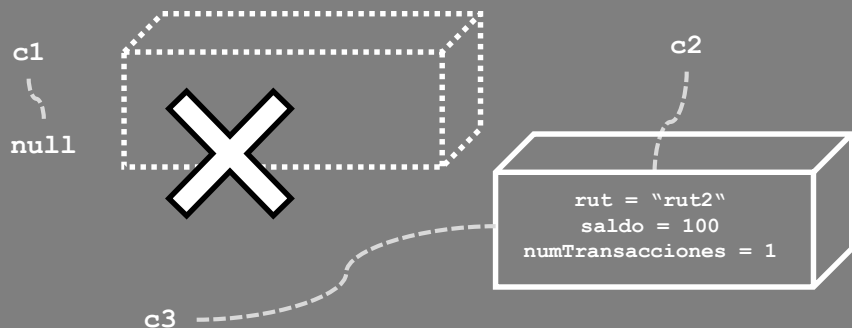
```
c1 = null;
System.out.println(c1.getSaldo());
```



Manejo de estado

Escriba una **nueva aplicación** en el mismo package donde está la clase **CuentaCorriente**, e incrementalmente vaya probando los siguientes códigos...

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente c1 = new CuentaCorriente("rut1");
c1.depositar(100);
c1.girar(10);
System.out.println(c1.getSaldo()); // 90
```

```
CuentaCorriente c2 = new CuentaCorriente("rut2");
c2.depositar(50);
c2.girar(70);
System.out.println(c2.getSaldo()); // 50
```

```
CuentaCorriente c3 = c2;
System.out.println(c3.getSaldo()); // 50

c3.depositar(50);
System.out.println(c2.getSaldo()); // 100
```

```
c1 = null;
System.out.println(c1.getSaldo());
```



Manejo de estado

Puede hacer el mismo ejercicio para un
arreglo de objetos de tipo `cuentaCorriente`





Manejo de estado

Puede hacer el mismo ejercicio para un **arreglo de objetos** de tipo `CuentaCorriente`

Memoria del computador

¿Qué imprime por pantalla?

```
CuentaCorriente cuentas = new CuentaCorriente[3];  
  
for(int i = 0; i < cuentas.length; i++)  
    System.out.println(cuentas[i]);
```



Manejo de estado

Puede hacer el mismo ejercicio para un **arreglo de objetos** de tipo `CuentaCorriente`

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente cuentas = new CuentaCorriente[3];  
  
for(int i = 0; i < cuentas.length; i++)  
    System.out.println(cuentas[i]); // null x 3
```



Manejo de estado

Puede hacer el mismo ejercicio para un **arreglo de objetos** de tipo `CuentaCorriente`

Memoria del computador



¿Qué imprime por pantalla?

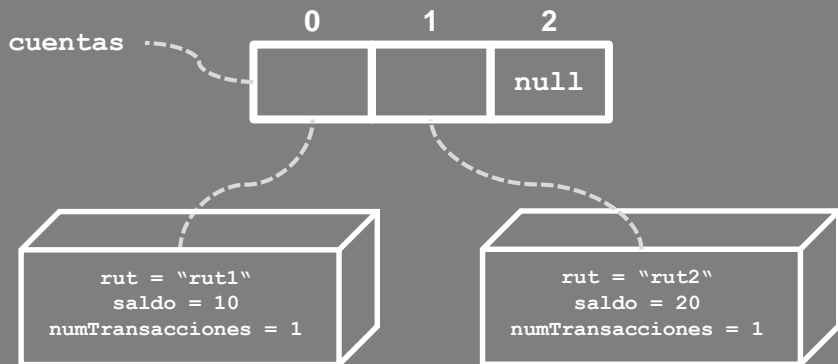
```
CuentaCorriente cuentas = new CuentaCorriente[3];  
  
for(int i = 0; i < cuentas.length; i++)  
    System.out.println(cuentas[i]); // null x 3  
  
cuentas[0] = new CuentaCorriente("rut1");  
cuentas[0].depositar(10);  
cuentas[1] = new CuentaCorriente("rut2");  
cuentas[1].depositar(20);  
  
for(int i = 0; i < cuentas.length; i++) {  
    if(cuentas[i] != null)  
        System.out.println(cuentas[i].getSaldo());  
}
```




Manejo de estado

Puede hacer el mismo ejercicio para un **arreglo de objetos** de tipo `CuentaCorriente`

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente cuentas = new CuentaCorriente[3];

for(int i = 0; i < cuentas.length; i++)
    System.out.println(cuentas[i]); // null x 3
```

```
cuentas[0] = new CuentaCorriente("rut1");
cuentas[0].depositar(10);
cuentas[1] = new CuentaCorriente("rut2");
cuentas[1].depositar(20);
```

```
for(int i = 0; i < cuentas.length; i++) {
    if(cuentas[i] != null)
        System.out.println(cuentas[i].getSaldo());
} // 10, 20
```

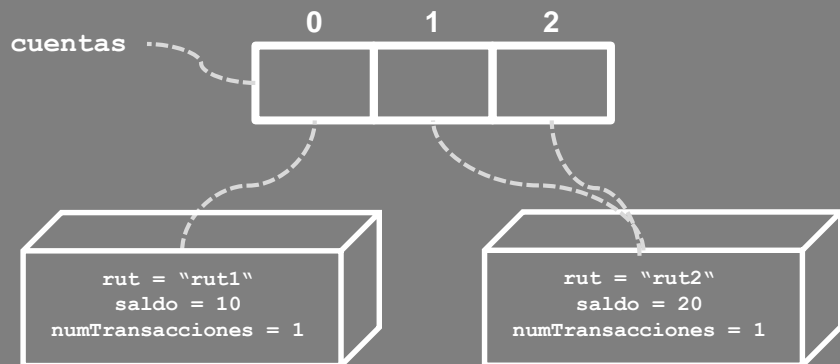
```
cuentas[2] = cuentas[1];
System.out.println(cuentas[2].getSaldo());
```




Manejo de estado

Puede hacer el mismo ejercicio para un **arreglo de objetos** de tipo `CuentaCorriente`

Memoria del computador



¿Qué imprime por pantalla?

```
CuentaCorriente cuentas = new CuentaCorriente[3];

for(int i = 0; i < cuentas.length; i++)
    System.out.println(cuentas[i]); // null x 3
```

```
cuentas[0] = new CuentaCorriente("rut1");
cuentas[0].depositar(10);
cuentas[1] = new CuentaCorriente("rut2");
cuentas[1].depositar(20);
```

```
for(int i = 0; i < cuentas.length; i++) {
    if(cuentas[i] != null)
        System.out.println(cuentas[i].getSaldo());
} // 10, 20
```

```
cuentas[2] = cuentas[1];
System.out.println(cuentas[2].getSaldo()); // 20
```



Manejo de estado

Actividad práctica

Desarrolle una aplicación en Java que maneje un arreglo de objetos `CuentaCorriente` de largo N (solicitado al usuario en el inicio), en donde se despliegue un menú que ofrezca las siguientes funciones:

- ☐ Agregar un nuevo cliente, validando que su RUT no se repita, y que haya espacio para ingresarlo.
- ☐ Depositar en una cuenta, buscándola por su RUT.
- ☐ Calcular el promedio de los saldos de las cuentas ingresadas.



Banco.java



Estructuras avanzadas

Más allá de arreglos



Universidad
de Aysén

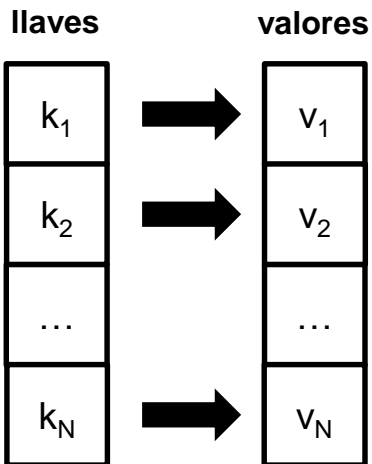


Estructuras avanzadas

- ☐ Los arreglos son útiles cuando
 - ☐ El **volumen** de los datos es **estimable**
 - ☐ Las **operaciones** que hay que hacer sobre dichos datos son **relativamente simples**
- ☐ Para **otros casos**, Java provee la librería **java.util**, en donde es posible encontrar **estructuras de datos avanzadas**, de las que veremos dos:
 - ☐ Listas
 - ☐ Mapas



**Implementadas
usando POO!**

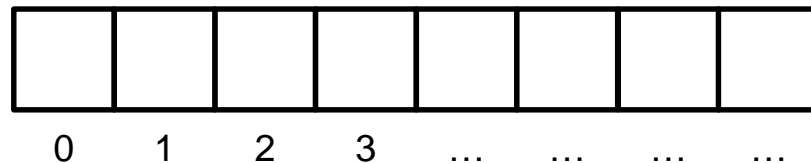


Mapas

Colección de duplas **llave-valor**, en donde es posible acceder a los valores a través de una llave única en la colección

Listas

Colección **indexada** de datos, que funciona internamente con un arreglo, cuyo tamaño se va ajustando **dinámicamente**





Uso de Listas

crear

```
ArrayList<CuentaCorriente> cuentas = new ArrayList<>();
```

agregar

```
cuentas.add(new CuentaCorriente("rut1"));
```

```
CuentaCorriente c = new CuentaCorriente("rut2");  
cuentas.add(c);
```

obtener

```
System.out.println(cuentas.get(0).getSaldo());
```

recorrer

```
for(int i = 0; i < cuentas.size(); i++)  
    System.out.println("Rut de cuenta: " + cuentas.get(i).getRut());
```

eliminar

```
cuentas.remove(2);  
cuentas.remove(c);
```

reemplazar

```
cuentas.set(0, new CuentaCorriente("rut5"));
```

Obtener cantidad

```
System.out.println("Tamaño: " + cuentas.size());
```

Borrar todo

```
cuentas.clear();
```



Uso de Mapas

crear

```
HashMap<String, CuentaCorriente> cuentas = new HashMap<>();
```

agregar

```
cuentas.put("rut1", new CuentaCorriente("rut1"));
```

```
CuentaCorriente c = new CuentaCorriente("rut2");  
cuentas.put("rut2", c);
```

obtener

```
System.out.println(cuentas.get("rut1").getSaldo());
```

recorrer

```
Iterator<String> keys = cuentas.keySet().iterator();  
  
while(keys.hasNext()) {  
    CuentaCorriente cuenta = cuentas.get(keys.next());  
    System.out.println("Rut " + cuenta.getRut());  
}
```

eliminar

```
cuentas.remove("rut");
```

Borrar todo

```
cuentas.clear();
```

Obtener cantidad

```
System.out.println("Tamaño: " + cuentas.size());
```



Estructuras avanzadas



ObrerosConstru.java

Ejemplo: Una empresa de construcción desea llevar el registro detallado de la asignación de trabajos que realizan los obreros contratados.

Para ello, se le ha pedido implementar la clase **Obrero**, que tiene el RUT del trabajador (String) y su edad (int).

Luego, debe implementar una aplicación con menú que maneje una lista de los obreros contratados. Al mismo tiempo, se debe poder asociar ID únicos de trabajos (int) a obreros, a través de un mapa.

El menú debe ofrecer las siguientes funcionalidades:

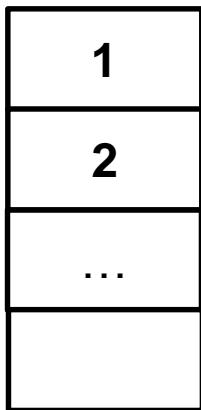
- ☐ Agregar un nuevo obrero, validando que su RUT no se repita.
- ☐ Asignar un ID de trabajo a un obrero existente, siempre y cuando dicho ID no este ya asignado
- ☐ Finalizar un trabajo con ID específico, es decir, quitar la asignación que pueda tener a un trabajador.
- ☐ Indicar el RUT del trabajador que está asignado a un ID de trabajo específico
- ☐ Calcular el “ratio de ociosidad”, es decir, la cantidad de obreros que no tienen un trabajo asignado, dividido por el total de obreros contratados.





Estructuras utilizadas:

Mapa de trabajos

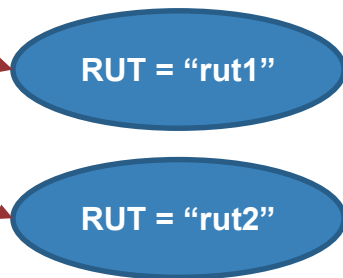


HashMap<Integer, Obrero> **asignaciones**

ArrayList<Obrero> **obreros**



Lista de obreros



Instancias de
Obrero



Estructuras avanzadas

Tips de búsqueda:

Buscar un obrero en la lista por su RUT

```
boolean encontrado = false;

for(int i = 0; i < obreros.size() && !encontrado; i++) {
    if(obreros.get(i).getRut().equals(nuevoRut))
        encontrado = true;
}

if(encontrado)
    System.out.println("Obrero ya existe");
```

```
System.out.println("Ingrese ID de trabajo a asignar");
int idTrabajo = lector.nextInt();

if(asignaciones.containsKey(idTrabajo))
    System.out.println("Trabajo ya está asignado");
```

Buscar si trabajo está asignado en el mapa

Programación Orientada a Objetos en Java

IN1015 – Programación II: Aplicaciones computacionales
Ingeniería Civil Industrial – Universidad de Aysén
Prof. Enrique Urra – enrique.urra@uaysen.cl



Universidad
de Aysén