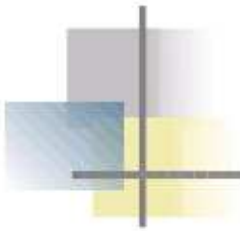
Programación en JAVA

Unidad 3
Clases y Objetos

Universidad de Chile
Departamento de Ciencias de la Computación
Profesor: Felipe Aguilera V.
faguiler@dcc.uchile.cl, felipe@aguilera.cl

Temario

- Clases
- Objetos
- Variables
- Métodos
- Sobrecarga de métodos
- Encapsulación
- Constructores
- Instanciación



OOP

- En Object-Oriented Programming (OOP), o Programación Orientada a Objetos, un programa está hecho de clases, con sus propiedades y operaciones

- La creación de un programa involucra ensamblar objetos y hacerlos comunicarse entre ellos (componer)

UML

- Unified Modeling Language es el lenguaje o notación estándar para Análisis y Diseño Orientados a Objetos
- Creado por Rational (http://www.rational.com), y mantenido por OMG (http://www.omg.org)
- En el curso, se usará UML para describir los conceptos introducidos

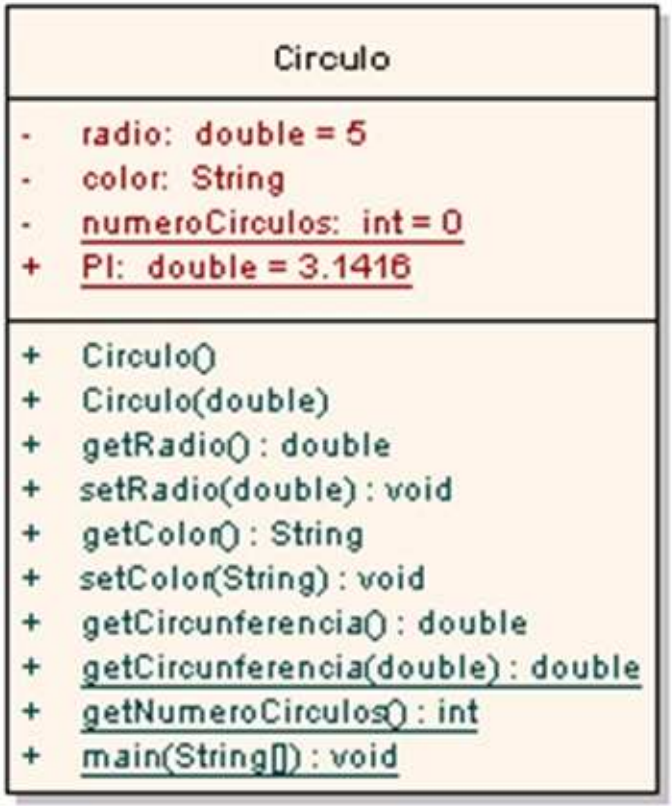
Clases y Objetos

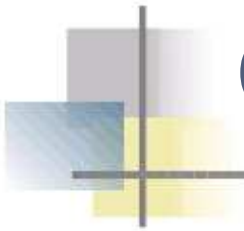
- Una clase describe un grupo de objetos que comparten propiedades y métodos comunes
- Una clase es una plantilla que define qué forma tienen los objetos de la clase
- Una clase se compone de:
  - Información: campos (atributos, propiedades)
  - Comportamiento: métodos (operaciones, funciones)
- Un objeto es una instancia de una clase

Clase	Objeto
Empresa	Sodimac
Casa	La Moneda
Empleado	Juan Pérez
Ventana (tiempo de diseño)	Ventana (tiempo de ejecución)
String	"Juan Pérez"

Definición de una Clase

```
class Circulo {
    // campos
    // métodos
    // constructores
    // main()
}
```





Campos

- Almacenamiento de información
  - Variables de instancia
  - Variables de clase (static)

Variable de Instancia

- Existe una instancia por cada objeto
- Puede ser inicializada en la declaración
- Una variable de instancia declarada final debe ser inicializada en la declaración (o en el constructor), y no puede ser modificada posteriormente

- Sintaxis: <tipo> <identificador> [= <valor inicial>];

- Ejemplo:

```
double radio = 5;
String color;
```

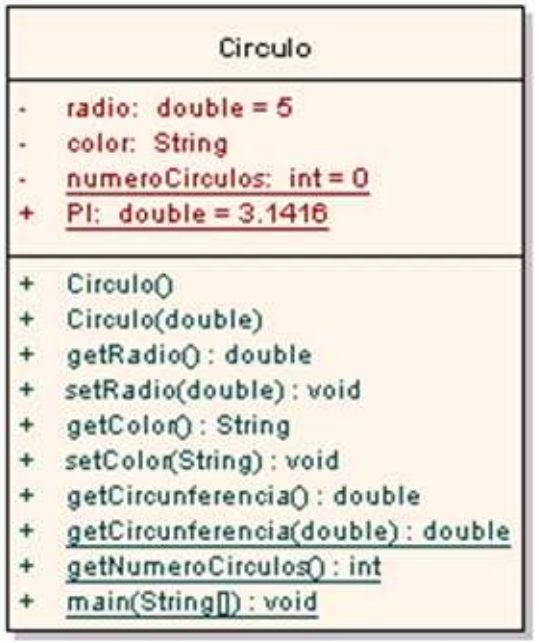
Variable de Clase (static)

- Una variable static existe una vez en memoria, independientemente del número de instancias de la clase
- Es accesible sin necesidad de instanciar la clase
- Puede ser inicializada en la declaración
- Una variable static declarada final debe ser inicializada en la declaración (o en el bloque de inicialización estática), y no puede ser modificada posteriormente
- Ejemplo:

```
static int numeroCirculos = 0;
static final double PI = 3.1416;
```

Campos en una Clase

```
class Circulo {
    // campos
    double radio = 5;
    String color;
    static int numeroCirculos = 0;
    static final double PI = 3.1416;
    // métodos
    // constructores
    // main( )
}
```



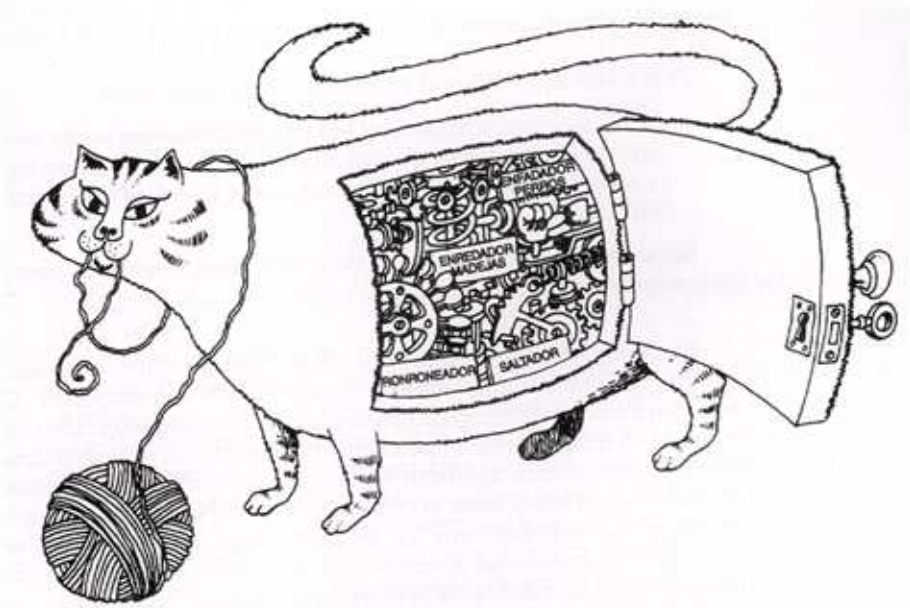
Acceso a Campos

- El acceso a miembros se realiza utilizando "dot notation"

```
Circulo c1 = new Circulo();
c1.radio = 5;
c1.color = "rojo";
Circulo.numeroCirculos++;
```

Encapsulación

- Ocultamiento de la implementación interna al usuario del objeto (cliente)



- Regla
  - Un objeto no debe acceder directamente a campos de otros objetos

Modificadores de Acceso

- Public: Accesible en cualquier lugar en que la clase sea accesible
- Protected: Accesible por subclases y clases del mismo package
- Package (default): Accesible por clases del mismo package
- Private: Accesible sólo al interior de la clase

Beneficios de la Encapsulación

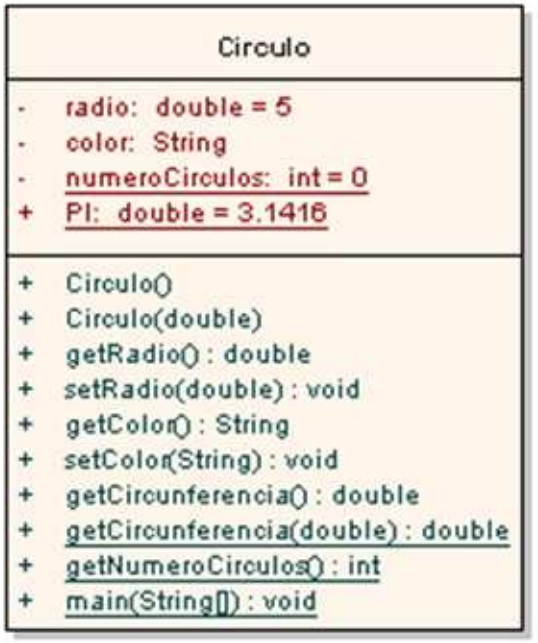
- La encapsulación da un nivel de seguridad al código, al restringir el acceso a la implementación

- La implementación interna puede ser modificada sin afectar el código de la aplicación - sólo hay impacto en los métodos de la clase

- Al proveer un método "setter" para modificar un campo privado, éste puede realizar chequeo de errores

Definición del Acceso

```
class Circulo {
    // campos
    private double radio = 5;
    private String color;
    private static int numeroCirculos = 0;
    public static final double PI = 3.1416;
    // métodos
    // constructores
    // main( )
}
```



Métodos

- Instrucciones que operan sobre los datos de un objeto para obtener resultados
- Tienen cero o más parámetros
- Pueden retornar un valor o pueden ser declarados void para indicar que no retornan ningún valor

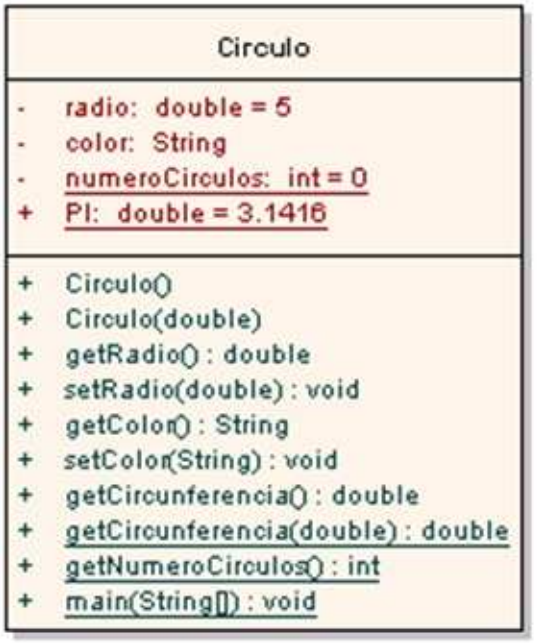
- Sintaxis

```
<tipo retorno> <nombre método> (<tipo> parámetro1, ...)
{
    // cuerpo del método
    return <valor de retorno>;
}
```

Método de Instancia

- Tiene acceso directo a las variables de instancia del objeto sobre el cual es invocado

```
double getCircunferencia()
{
  return 2 * radio * PI;
}
```

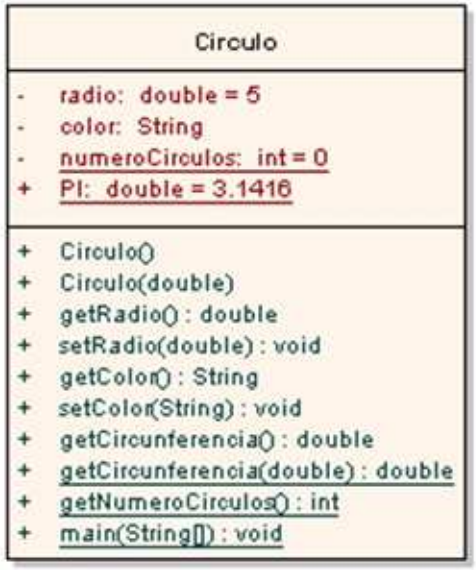


- Es invocado sobre un objeto de la clase

```
Circulo c = new Circulo();
c.setRadio(20);
double d = c.getCircunferencia();
```

Método de Clase (static)

- No actúa sobre ninguna instancia de la clase
- Puede ser utilizado sin instanciar la clase
- Sólo tiene acceso directo a variables static de la clase




```
public static double getCircunferencia(double r) {
return 2 * r * PI;
}
```

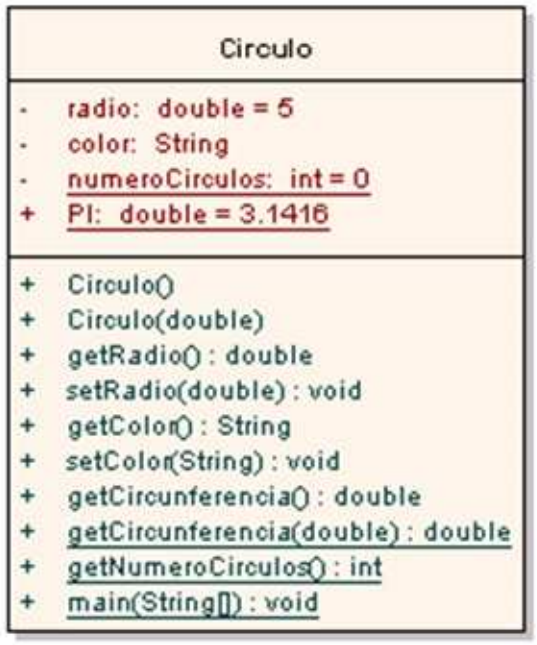
- Es invocado directamente sobre la clase

```
double d = Circulo.getCircunferencia(30);
```

Definición de Métodos

```
class Circulo {
    // campos

    // métodos
    public double getRadio() {...}
    public void setRadio(double radio) {...}
    public String getColor() {...}
    public void setColor(String color) {...}
    public double getCircunferencia() {...}
    public static double getCircunferencia(double radio) {...}
    public static int getNumeroCirculos() {...}
}
```



Sobrecarga (overload) de Métodos

- Métodos de una clase pueden tener el mismo nombre pero diferentes parámetros
- Cuando se invoca un método, el compilador compara el número y tipo de los parámetros y determina qué método debe invocar
- Firma (signature) = nombre del método + lista de parámetros
- Ejemplo:

```
class Cuenta {
    public void depositar(double monto) {
        this.depositar(monto, "$");
    }
    public void depositar(double monto,
                          String moneda) {
        // procesa el depósito
    }
}
```

Constructores

- Un constructor es un método especial invocado para instanciar e inicializar un objeto de una clase
  - Invocado con la sentencia new
  - Tiene el mismo nombre que la clase
  - Puede tener cero o más parámetros
  - No tiene tipo de retorno, ni siquiera void
  - Un constructor no público restringe el acceso a la creación de objetos

Constructor Default

- Es un constructor sin parámetros creado por el compilador si uno no provee ningún constructor

- Si la clase tiene algún constructor, entonces el constructor sin parámetros debe ser explícitamente creado en caso que se requiera

Ejemplo

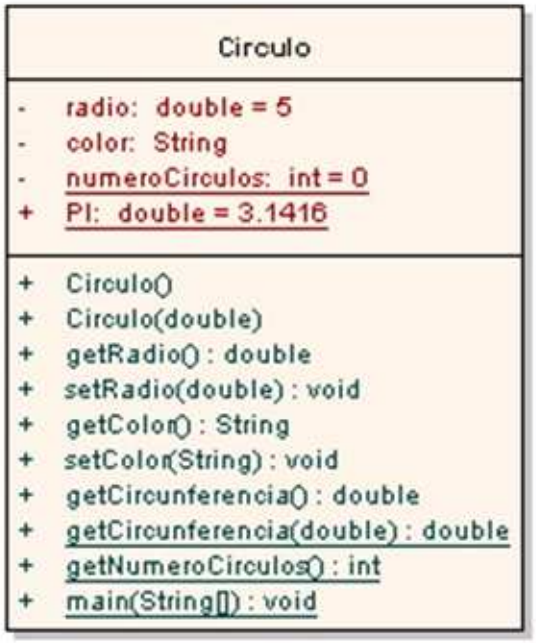
```
class Circulo {


    // constructores
    public Circulo() {
    }

    public Circulo(double r) {
        radio = r;
    }

    void f() {
        Circulo c = new Circulo(30);

    }
}
```



Inicialización Static

- Es posible definir un bloque de inicialización static al interior de una clase, que se ejecuta en la primera ocasión en que la máquina virtual realiza una referencia a la clase

```
public class Log {
    private static DataSource datasource;
    static {
        try {
            datasource = ...;
        } catch (Exception e) {

        }
    }
```

El Método main

- Punto de partida de la aplicación
- No es necesario si la clase no es usada como punto de partida de la aplicación
- Es invocado sin crear una instancia (static)

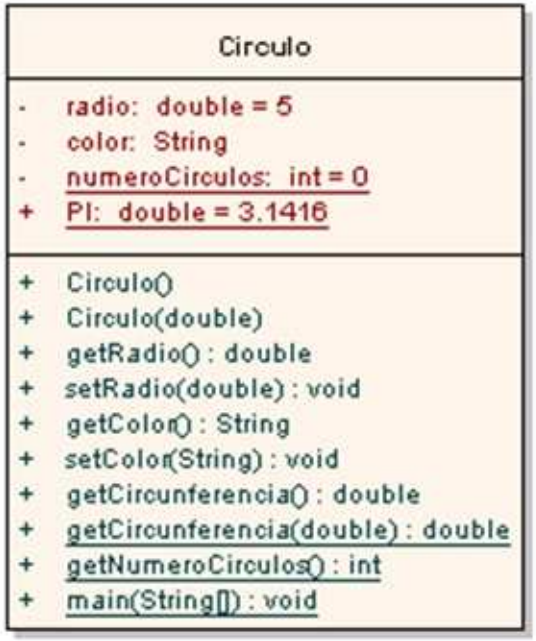
Firma de main

- El método main debe:
  - Tener acceso public
  - Ser de tipo static
  - Retornar void
  - Recibir un parámetro String[]

Definición del Método main

```
class Circulo {


    // método main()
    public static void main(String args[]) {
        Circulo c = new Circulo();
    }
}
```



Declaración de Variables

- La definición de una clase crea un tipo de datos
- Variables de este tipo se declaran:

```
Circulo c1;
```

- Una declaración no crea un objeto; crea una variable que contiene una referencia a un objeto, sin crear al objeto en sí

Instanciación

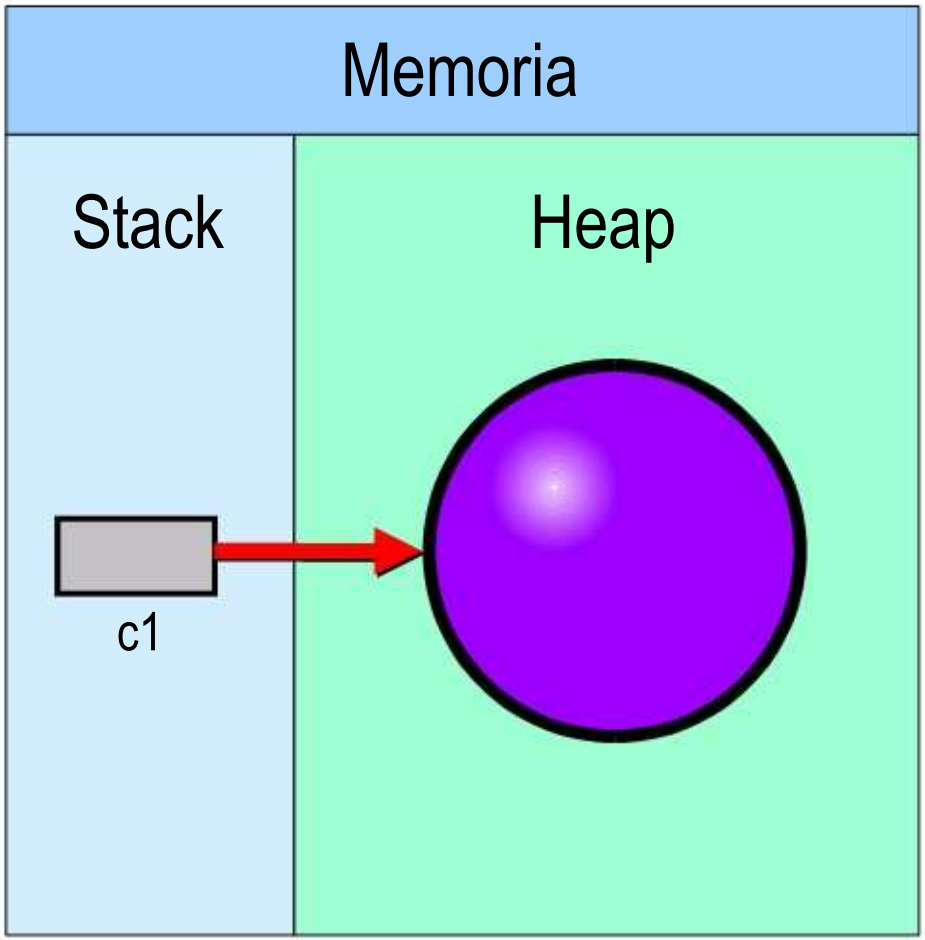
- Los objetos se crean usando el operador new

```
c1 = new Circulo();
```

- Los objetos son creados en un área de memoria conocida como el heap
- Todos los objetos son utilizados vía referencias

Instanciación

```
Circulo c1;
c1 = new Circulo();
```



Uso de Miembros de Instancia

- Para acceder a variables y métodos de instancia se utiliza la sintaxis "objeto."

```
Circulo c1 = new Circulo();
c1.radio = 5;
c1.color = "rojo";
double d = c1.getCircunferencia();
```

- Si la referencia es null, se genera una excepción NullPointerException

Uso de Miembros de Clase

- Para acceder a variables y métodos static se utiliza la sintaxis "clase."

```
Circulo c1 = new Circulo();
Circulo.numeroCirculos++;
int n = Circulo.getNumeroCirculos();
double d = Circulo.PI;
```

Uso de this

- En un método de instancia, this es una referencia al objeto sobre el cual se invocó el método

```
class Circulo {

    void grabar() {

        Database.save(this);
        LogManager.log(this);
    }

}
```

Uso de this

- La palabra this puede ser utilizada en la primera línea de un constructor para invocar a otro constructor

```
class Circulo {
    private double radio;
    private static int numeroCirculos = 0;
    Circulo(double radio) {
        this.radio = radio;
        Circulo.numeroCirculos++;
    }
    Circulo() {
        this(10);   // radio default: 10
    }
}
```

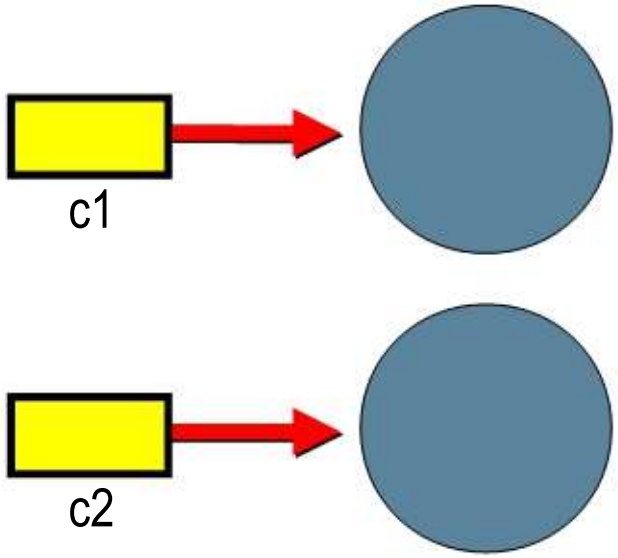
Garbage Collection

- Cuando no quedan en el programa referencias a un objeto, el espacio que él ocupa puede ser reclamado por un "garbage collector" (recolector de basura)
- Uno no elimina explícitamente objetos (no existe el delete de C++)
- Entorno seguro (no hay punteros a basura)
- Es posible invocar directamente al garbage collector para intentar forzar la recolección de basura:

```
System.gc();
```

Garbage Collection

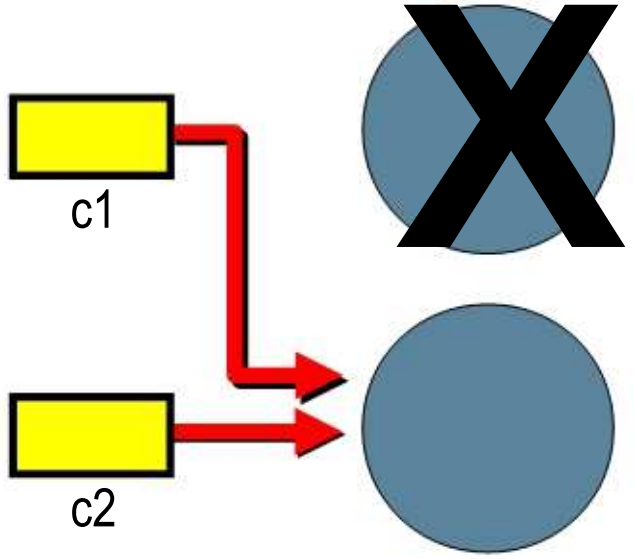
```
Circulo c1 = new Circulo();
Circulo c2 = new Circulo();
c1 = c2;
```



Garbage Collection

```
Circulo c1 = new Circulo();
Circulo c2 = new Circulo();
c1 = c2;
```

- Nota: el primer círculo no es eliminado de memoria al perderse su referencia, sino cuando la JVM decide invocar al garbage collector



Resumen

- Una clase es una plantilla a partir de la cual se instancian objetos
- Los objetos contienen información (en variables de instancia y de clase) y comportamiento (en métodos de instancia y de clase)
- Los miembros de instancia se utilizan con la sintaxis "objeto."
- Los miembros de clase (static) se utilizan con la sintaxis "clase."
- Una clase puede tener varios métodos con el mismo nombre (sobrecarga), siempre que tengan diferentes parámetros

Resumen

- La ejecución de un programa comienza en el método main() de una clase
- Para instanciar una clase (crear un objeto) se utiliza el operador new
- Los constructores son métodos especiales invocados al instanciar una clase
- Los objetos se manejan a través de referencias
- La palabra this representa una referencia al objeto sobre el cual se invoca un método de instancia
- Los modificadores de acceso controlan quién tiene acceso a los campos y métodos de una clase