Sesión 05: Terminología

Programación 2

Ángel Herranz

Febrero 2019

Universidad Politécnica de Madrid

En capítulos anteriores

- Sobre los IDEs
- Clases y objetos (intro)
- Objetos, referencias y variables (y primitivos)
- Clases: plantilla para crear objetos

En capítulos anteriores

- Sobre los IDEs
- Clases y objetos (intro)
- Objetos, referencias y variables (y primitivos)
- Clases: plantilla para crear objetos

Encapsular

datos y comportamiento

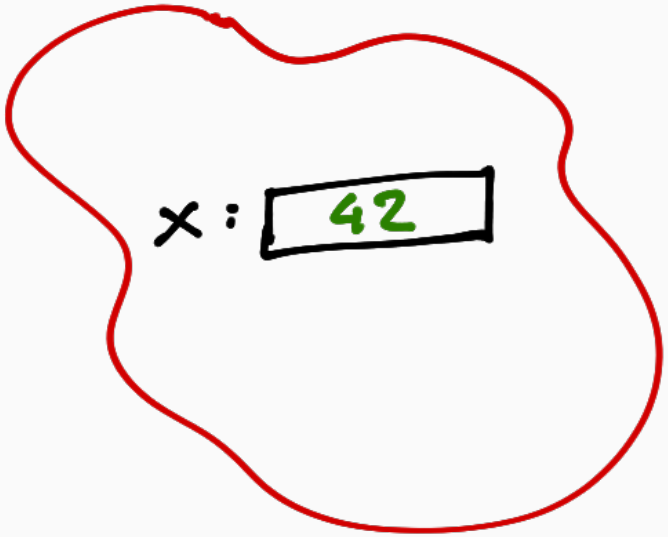
- Racionales

Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```

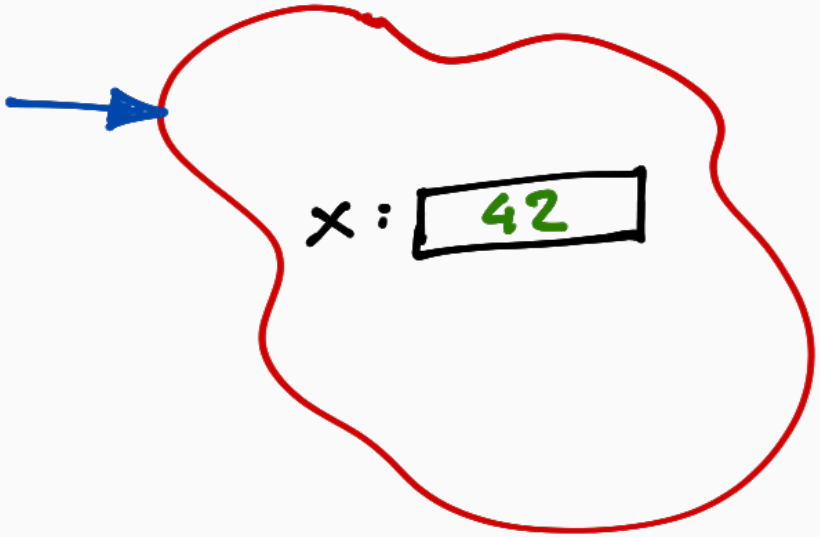
Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



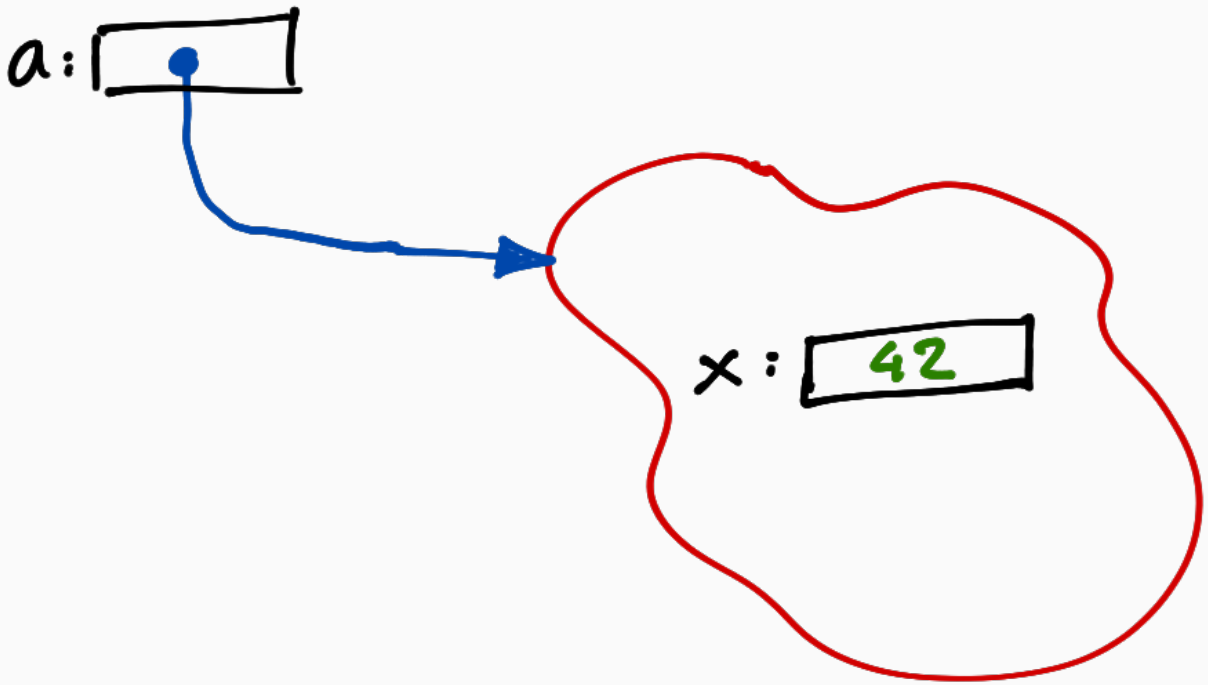
Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



Clases: datos + comportamiento

```
class Cancion {
  String titulo;
  String interprete;
  int duracion;
  ...
  Cancion (String titulo,
           String interprete,
           int duracion) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duracion;
  }
  ...
  String duracion() {
    String minutos =
      duracion / 60;
    String segundos =
      duracion % 60;
    if (segundos.length() == 1)
      segundos
        = "0" + segundos;
    }
    return
      minutos + ":" + segundos;
  }
}
```

Clases: datos + comportamiento

```
class Cancion {
  String titulo;
  String interprete;
  int duracion;
  ...
  Cancion (String titulo,
           String interprete,
           int duracion) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duracion;
  }
  ...
  String duracion() {
    String minutos =
      duracion / 60;
    String segundos =
      duracion % 60;
    if (segundos.length() == 1)
      segundos
        = "0" + segundos;
    }
    return
      minutos + ":" + segundos;
  }
}
```

Clases: datos + comportamiento

```
class Cancion {
  String titulo;
  String interprete;
  int duracion;
  ...
  Cancion (String titulo,
           String interprete,
           int duracion) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duracion;
  }
  ...
  String duracion() {
    String minutos =
      duracion / 60;
    String segundos =
      duracion % 60;
    if (segundos.length() == 1)
      segundos
        = "0" + segundos;
    }
    return
      minutos + ":" + segundos;
  }
}
```

Clases: datos + comportamiento

```
class Cancion {
  String titulo;
  String interprete;
  int duracion;
  ...
  Cancion (String titulo,
           String interprete,
           int duracion) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duracion;
  }
  ...
  String duracion() {
    String minutos =
      duracion / 60;
    String segundos =
      duracion % 60;
    if (segundos.length() == 1)
      segundos
        = "0" + segundos;
    }
    return
      minutos + ":" + segundos;
  }
}
```

Clases: datos + comportamiento

```
class Cancion {
  String titulo;
  String interprete;
  int duracion;
  ...
  Cancion (String titulo,
           String interprete,
           int duracion) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duracion;
  }
  ...
  String duracion() {
    String minutos =
      duracion / 60;
    String segundos =
      duracion % 60;
    if (segundos.length() == 1)
      segundos
        = "0" + segundos;
    }
    return
      minutos + ":" + segundos;
  }
}
```

Terminología OO + Java i

atributo, clase, constructor, dato
básico, instancia, invocación, mensaje,
miembro, método, modificador,
objeto, observador, puntero,
referencia, tipo, variable

Terminología OO + Java i

attribute, class, constructor, basic
data, instance, invocation, message,
member, method, modifier,
object, observer, pointer,
reference, type, variable

Datos y comportamiento

- Nadie usa los términos *datos* y *comportamiento*
- Datos: atributos (*attribute*)
- Comportamiento: métodos (*method*)
  - Constructor: mismo nombre que la clase
  - Observador: observa, devuelve algo, no modifica
  - Modificador modifica, no devuelve nada (**`void`**)

- Ámbos (datos y métodos): miembros (*member*)[2]

[2] Terminología Java

Datos y comportamiento

- Nadie usa los términos *datos* y *comportamiento*
- Datos: atributos (*attribute*)
- Comportamiento: métodos (*method*)
  - Constructor: mismo nombre que la clase
  - Observador: observa, devuelve algo, no modifica
  - Modificador modifica, no devuelve nada (**`void`**)
  - ¿Puede haber métodos *observamodificadores*?[1]
- Ámbos (datos y métodos): miembros (*member*)[2]

[1] Sí, pero hagamos caso a Bertrand Meyer y a su principio CQS (*Command Query Separation Principle*)

[2] Terminología Java

Algunos sinónimos

$$\begin{aligned} \text{Objeto} &= \text{Instancia} \\ \text{Puntero} &= \text{Referencia} \\ \text{Método} &= \text{Mensaje} \\ \text{Invocar} &= \text{Enviar mensaje} \\ \text{Clase} &\subseteq \text{Tipo} \end{aligned}$$

recordar Cuando usas un código

- No quieres saber cómo está hecho
- Sólo quieres saber qué hace
- No quieres repetir trabajo

Ocultación i

- Estás usando `Racional`

```
System.out.println(
  "El valor decimal es " + r.num / r.den
);
```

- El desarrollador de `Racional` decide renombrar `num` y `den` por `numerador` y `denominador`

Ocultación i

¿Qué va a pasar?

- Estás usando `Racional`

```
System.out.println(
  "El valor decimal es " + r.num / r.den
);
```

- El desarrollador de `Racional` decide renombrar `num` y `den` por `numerador` y `denominador`

Ocultación ii

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {
  double x;
  double y;
  ...
}
```

- Tienes un modificador `void rotar(double rad)`

Ocultación ii

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {
  double x;
  double y;
  ...
}
```

- Tienes un modificador `void rotar(double rad)`
- Y ahora sabes que sabes que va a ser usado masivamente

Ocultación ii

¿Qué vas a hacer?

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {
  double x;
  double y;
  ...
}
```

- Tienes un modificador `void rotar(double rad)`
- Y ahora sabes que sabes que va a ser usado masivamente

Ocultación ii

¿Qué vas a hacer?

¿Qué va a pasar?

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {
  double x;
  double y;
  ...
}
```

- Tienes un modificador `void rotar(double rad)`
- Y ahora sabes que sabes que va a ser usado masivamente

¿Ideas?

¿Qué vas a hacer?

- Utilizar coordenadas polares para representar los puntos

```
class Punto2D {
  double radio;
  double angulo;
```

- Implementar rotar con muy poco esfuerzo:

```
  void rotar(double rad) {
    angulo = angulo + rad;
  }
}
```

¿Qué va a pasar?

- Cualquier otro método implementado en la clase dejará de funcionar
- Al modificar los atributos, todo el código que usa `Punto2D` dejará de compilar

Código “cliente”

```
public static void main(String[] args) {
  Punto2D a;
  Punto2D b;
  // Quiero que new Punto2D() cree una
  // instancia que represente el (0,0).
  a = new Punto2D();
  // Quiero que new Punto2D(x,y) cree una
  // instancia que represente el (x,y).
  b = new Punto2D(1,-1);
  // Voy a calcular la distancia en los dos
  // puntos
  double d = a.distancia(b);
  // Compruebo que está bien calculada
  System.out.println(Math.sqrt(2) == d);
}
```

Punto2D

1. Programar `Punto2D` usando coordenadas cartesianas
2. Compilar y ejecutar
3. Reescribir `Punto2D` usando coordenadas polares
4. Compilar y ejecutar

`private i`

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, es muy importante ocultar dicha representación

private i

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, es muy importante ocultar dicha representación

```
class Punto2D {
          double x;
          double y;
   ...
}
```

```
public static void
   main(String[] args) {
   Punto2D p =
      new Punto2D();
   p.x = 1;
   p.y = -1;
}
```

private i

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, es muy importante ocultar dicha representación

```
class Punto2D {
  private double x;
  private double y;
  ...
}
```

```
public static void
  main(String[] args) {
  Punto2D p =
    new Punto2D();
  p.x = 1;👎 No compila
  p.y = -1;👎 No compila
}
```

Buenas prácticas i

En lugar de exponer los atributos se define un API de acceso a la clase

- `Punto2D()`: crea un punto $(0, 0)$ en coordenadas cartesianas
- `Punto2D(x,y)`: crea un punto (x, y) en coordenadas cartesianas (abscisa y ordenada)
- `x()`: devuelve la abscisa
- `y()`: devuelve la ordenada

Buenas prácticas ii

- `cambiarX(double x)`: modifica la abscisa
- `cambiarY(double y)`: modifica la ordenada
- `distancia(Punto2D b)`: devuelve la distancia desde **`this`** a `b`
- `rotar(double a)`: rota el punto alrededor del $(0, 0)$ la cantidad de `a` radianes

Buenas prácticas ii

- `cambiarX(double x)`: modifica la abscisa
- `cambiarY(double y)`: modifica la ordenada
- `distancia(Punto2D b)`: devuelve la distancia desde **`this`** a `b`
- `rotar(double a)`: rota el punto alrededor del $(0, 0)$ la cantidad de **`a`** radianes

¡Implementar la buena práctica!