

# Sesión 11: Tipos Abstractos de Datos

## Programación 2

### 4. Tipos Abstractos de Datos

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

# En capítulos anteriores



Tema 1: Clases y Objetos



Tema 2: Colecciones acotadas de Objetos

# En capítulos anteriores



Tema 1: Clases y Objetos



Tema 2: Colecciones acotadas de Objetos

- Todo con *arrays* *nativos*:  $T[]$
- *Nulles*
- Índices
- Pilas *acotadas*

# En el capítulo de hoy

 Tema 2: Colecciones acotadas de Objetos

# En el capítulo de hoy

- 👍 Tema 2: Colecciones acotadas de Objetos
  - 🏠 “Listas” (hoja de ejercicios)

# En el capítulo de hoy

- 👍 Tema 2: Colecciones acotadas de Objetos
  - 🏠 “Listas” (hoja de ejercicios)
- 🕒 Tema 3: Programación modular
- 👍 Tema 4: Tipos Abstractos de Datos

# En el capítulo de hoy

 Tema 2: Colecciones acotadas de Objetos

 “Listas” (hoja de ejercicios)

 Tema 3: Programación modular

 Tema 4: Tipos Abstractos de Datos

 *SIMs*



# Dos roles para ver un código

Cuando lo implementas

y

Cuando lo usas



# SIMs

- La idea es hacer un *simulador social*
- Las personas (SIMs) tienen un **nombre** (añade otras características, como **género**)
- Cada SIM puede **dormir, comer, trabajar, jugar, hablar, ...**
- Cada SIM puede hacerse **amigo** de otro SIM
- El simulador hace pasar el tiempo en pulsos de **una hora**

# Cuando usas un código. . .

Así ve la clase SIM el programador del simulador

# Cuando usas un código...

Así ve la clase SIM el programador del simulador

```
public class SIM {  
    public enum Actividad { DORMIR, COMER, ... };  
    public SIM(String nombre) ...  
    public void simular(int tiempoHoras) ...  
    public void hacerAmigo(SIM amigo) ...  
    public String nombre() ...  
    public Actividad quease() ...  
    public int estadistica(Actividad a) ...  
    public SIM amigo() ...  
}
```

... sólo ves su API

TAD

Tipo Abstracto de Datos

... sólo ves su API

TAD

Tipo Abstracto de Datos  
*Abstract Data Type (ADT)*

# ... sólo ves su API

## TAD

Tipo Abstracto de Datos  
*Abstract Data Type (ADT)*

- No quieres saber cómo está hecho
- Sólo quieres saber qué hace: semántica

# ... sólo ves su API

## TAD

Tipo Abstracto de Datos  
*Abstract Data Type (ADT)*

- No quieres saber cómo está hecho
  - Sólo quieres saber qué hace: semántica
- 🔊 ¿Entiendes la semántica de cada método?

# ... sólo ves su API

## TAD

Tipo Abstracto de Datos  
*Abstract Data Type (ADT)*

- No quieres saber **cómo** está hecho
- Sólo quieres saber **qué** hace: semántica
- 🎧 ¿Entiendes la semántica de cada método?
- 🎧 ¿Y la semántica *del todo*?



# Cuando implementas un código...

Así ve la clase SIM el implementador de SIM.java

```
import java.util.Random;

public class SIM {

    public enum Actividad {
        DORMIR, COMER, ESTUDIAR, JUGAR
    };

    private String nombre;
    private Actividad haciendo;
    private int horas;
    private SIM amigo;
    private int[] dedicado;

    public SIM(String nombre) {
        this.nombre = nombre;
        haciendo = Actividad.DORMIR;
        horas = 0;
        amigo = null;
        int n = Actividad.values().length;
        dedicado = new int[n];
    }

    public void simular(int tiempoHoras) {
        Actividad siguiente = null;
        horas = horas + tiempoHoras;
        switch (haciendo) {
            case DORMIR:
                if (horas < 8)
                    siguiente = Actividad.DORMIR;
                else
                    siguiente = Actividad.COMER;
                break;
            case ESTUDIAR:
                if (horas < 8)
                    siguiente = Actividad.ESTUDIAR;
                break;
            default:
                break;
        }
        ...
    }
}
```

# Cuando implementas un código...

Así ve la clase SIM el implementador de SIM.java

```
import java.util.Random;

public class SIM {

    public enum Actividad {
        DORMIR, COMER, ESTUDIAR, JUGAR
    };

    private String nombre;
    private Actividad haciendo;
    private int horas;
    private SIM amigo;
    private int[] dedicado;

    public SIM(String nombre) {
        this.nombre = nombre;
        haciendo = Actividad.DORMIR;
        horas = 0;
        amigo = null;
        int n = Actividad.values().length;
        dedicado = new int[n];
    }

    public void simular(int tiempoHoras) {
        Actividad siguiente = null;
        horas = horas + tiempoHoras;
        switch (haciendo) {
            case DORMIR:
                if (horas < 8)
                    siguiente = Actividad.DORMIR;
                else
                    siguiente = Actividad.COMER;
                break;
            case ESTUDIAR:
                if (horas < 8)
                    siguiente = Actividad.ESTUDIAR;
                break;
            default:
                break;
        }
        ...
    }
}
```

... sólo te preocupa el **cómo**

Modelización  
Estructuras de Datos

# ... sólo te preocupa el cómo

## Modelización

### Estructuras de Datos

- Estás al servicio de quienes lo usan
- para que no necesiten mirar el cómo

# ... sólo te preocupa el cómo

## Modelización

### Estructuras de Datos

- Estás al servicio de quienes lo usan
- para que no necesiten mirar el cómo



¿Entiendes la modelización?



¿Para qué está cada atributo?

¿Cómo explicamos la semántica?

---

<sup>1</sup>Quizás necesites la opción `-charset utf8` y para más tarde `-author`

# ¿Y el **qué**?

## ¿Cómo explicamos la semántica?

- Por el momento, sólo con comentarios
- Pero vamos a *javadococumentar*<sup>1</sup>

```
C:\ Sesion11> javadoc -d doc_sim SIM.java
Loading source file SIM.java...
...
Generating doc_sim/help-doc.html...
```

---

<sup>1</sup>Quizás necesites la opción `-charset utf8` y para más tarde `-author`

# Javadoc i

```
/**  
 * Las instancias de SIM representan seres humanos en una simulación  
 * extremadamente simplista.  
 *  
 * @author Ángel Herranz  
 */  
public class SIM {  
    ...  
}
```



# Javadoc ii

```
/**
 * Simula el paso del tiempo para este SIM. El SIM, con el paso del
 * tiempo, cambia de actividad y acumula la información de la
 * actividad anterior.
 *
 * @param tiempoHoras tiempo simulado (en horas)
 */
public void simular(int tiempoHoras) {
    ...
}
```

# Javadoc iii

```
/**  
 * Informa sobre el tiempo dedicado a la actividad indicada.  
 *  
 * @param a actividad sobre la que se quiere conocer el tiempo  
 * dedicado  
 * @return número de horas dedicadas a la actividad indicada  
 */  
public int estadistica(Actividad a) {  
    ...  
}
```

## Class SIM

java.lang.Object  
SIM

```
public class SIM
extends java.lang.Object
```

Las instancias de SIM representan seres humanos en una simulación extremadamente simplista.

**Author:**  
Ángel Herranz

### Nested Class Summary

#### Nested Classes

Modifier and Type	Class and Description
static class	SIM.Actividad

### Constructor Summary

#### Constructors

Constructor and Description
SIM(java.lang.String nombre)

### Method Summary

#### All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
SIM	amigo()
int	estadistica(SIM.Actividad a) Informa sobre el tiempo dedicado a la actividad indicada.
void	hacerAmigo(SIM amigo)
java.lang.String	nombre()
SIM.Actividad	quease()
void	simular(int tiempoHoras) Simula el paso del tiempo para este SIM.

# Entonces ¿qué es un TAD?

Un TAD (tipo abstracto de datos) lo definen

- Un **nombre**, el nombre del tipo.
- Unas **operaciones**.
- La **semántica** de las operaciones.

# Entonces ¿qué es un TAD?

Un TAD (tipo abstracto de datos) lo definen

- Un **nombre**, el nombre del tipo.
- Unas **operaciones**.
- La **semántica** de las operaciones.
- El mismo TAD puede tener **múltiples implementaciones concretas: estructuras de datos**.

# *Test Driven Development (TDD)*

Desarrollo dirigido por los tests

# *Test Driven Development (TDD)*

Desarrollo dirigido por los tests ⓘ

# *Test Driven Development (TDD)*

Desarrollo dirigido por los tests ?

Antes de implementar nada,  
implementamos los tests



# Test Driven Development (TDD)

Desarrollo dirigido por los tests ?

Antes de implementar nada,  
implementamos los tests



Lo aplicamos a este ejemplo

1. Escribimos el código de `SIM.java` “vacío”, sólo para que compile
2. Escribimos un programa principal `TestSIM.java` que compruebe propiedades de acuerdo a la semántica

# SIM.java “vacío”

```
public class SIM {

    public enum Actividad {
        DORMIR, COMER, ESTUDIAR, JUGAR
    };

    public SIM(String nombre) {
    }

    public void simular(int tiempoHoras) {
    }

    public String nombre() {
        return "Ángel";
    }

    public void hacerAmigo(SIM amigo) {
    }

    public SIM amigo() {
        return null;
    }

    public Actividad quease() {
        return Actividad.DORMIR;
    }

    public int estadistica(Actividad a) {
        return 0;
    }
}
```



# TestSIM.java

```
public class TestSIM {  
    public static void main(String[] args) {  
        SIM lucia = new SIM("Lucía");  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        lucia.simular(1);  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        assert "Lucía".equals(lucia.nombre());  
        lucia.simular(8);  
        assert lucia.quease() == SIM.Actividad.COMER;  
    }  
}
```



# TestSIM.java

```
public class TestSIM {  
    public static void main(String[] args) {  
        SIM lucia = new SIM("Lucía");  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        lucia.simular(1);  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        assert "Lucía".equals(lucia.nombre());  
        lucia.simular(8);  
        assert lucia.quease() == SIM.Actividad.COMER;  
    }  
}
```



## Compilar y ejecutar



## TestSIM.java

```
public class TestSIM {  
    public static void main(String[] args) {  
        SIM lucia = new SIM("Lucía");  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        lucia.simular(1);  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        assert "Lucía".equals(lucia.nombre());  
        lucia.simular(8);  
        assert lucia.quease() == SIM.Actividad.COMER;  
    }  
}
```



Compilar y ejecutar con `java -ea TestSIM`



# TestSIM.java

```
public class TestSIM {  
    public static void main(String[] args) {  
        SIM lucia = new SIM("Lucía");  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        lucia.simular(1);  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        assert "Lucía".equals(lucia.nombre());  
        lucia.simular(8);  
        assert lucia.quease() == SIM.Actividad.COMER;  
    }  
}
```



Compilar y ejecutar con `java -ea TestSIM`



Escribir más tests

## TestSIM.java

```
public class TestSIM {  
    public static void main(String[] ags) {  
        SIM lucia = new SIM("Lucía");  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        lucia.simular(1);  
        assert lucia.quease() == SIM.Actividad.DORMIR;  
        assert "Lucía".equals(lucia.nombre());  
        lucia.simular(8);  
        assert lucia.quease() == SIM.Actividad.COMER;  
    }  
}
```

 Compilar y ejecutar con `java -ea TestSIM`

 Escribir más tests **antes de programar** SIM.java

# Tecnología para hacer *TDD*: **assert**

- Ya veremos **tecnología más elaborada** para hacer testing (ej. **JUnit**)
- **De momento** usamos el **assert** de Java:<sup>2</sup>

**assert** *Condición* [*: Mensaje*];

- **Ejemplos** de uso:

```
assert lucia.quease() == SIM.Actividad.DORMIR  
    : "lucia debería estar durmiendo";
```

```
assert "Lucía".equals(lucia.nombre())  
    : "Nombre de lucia incorrecto: " + lucia.nombre();
```

---

<sup>2</sup>Los corchetes indican que el mensaje es opcional