Bloque 2

Introducción al diseño de tipos

Objetos, interfaces, clases

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

El tipo Object (*Ejercicio: Persona*)

2023/24

Corregimos que quedó pendiente.

Para el tipo ***Persona*** . Se trata ahora de implementar y probar los métodos ***toString()***, ***equals()*** y ***hashCode()***

Decidimos que:

- La representación textual de una persona es su dni, apellidos y nombre separador por punto y coma y un blanco y entre paréntesis:

  (12345678A; García Gómez; Ana)

- Dos Personas son iguales si tienen el ***mismo dni, nombre y apellidos***

Criterio de orden natural (Persona)

Corregimos que quedó pendiente.

Orden natural de Persona

Diremos que el criterio de orden natural del tipo Persona es por el dni

El tipo Object (*Ejercicio: Animal*)

2023/24

Corregimos que quedó pendiente.

Para el tipo *Animal* . Se trata ahora de implementar y probar los métodos *toString()*, *equals()* y *hashCode()*

Decidimos que:

- La representación textual de un animal es su nombre seguido de la familia entre corchetes:

  Buitre[AVE]

- Dos animales son iguales si tienen el *mismo nombre y familia*

Criterio de orden natural (Animal)

Corregimos que quedó pendiente.

Orden natural de Animal

Diremos que el criterio de orden natural del tipo Animal es por el pesoMedio y a igualdad de pesoMedio por edadMedia y a igualdad de edadMedia por la familia

Creación de tipos mediante record

2023/24

Inplementación de otros constructores en un record.

Hemos visto que una forma "minimalista" de crear un tipo es usando records, que proporciona un *constructor* que denominamos *canónico* que tiene un parámetro por cada atributo.

Pero ¿*y si queremos un constructor con menos parámetros*?. La solución es sencilla: Se construye un constructor en el record con los parámetros deseados que tiene *una primera línea* con la sentencia this(…) que invoca al constructor canónico.

Constructor de Persona con sólo apellidos y nombre

```
public record Persona(…){
   …
   public Persona(String apellidos, String nombre) {
      this("", nombre, apellidos, LocalDate.now());
   }
   …
}
```

El tipo Animal. Constructores

2023/24

Para el tipo *Animal* . Se trata ahora de *implementar y probar dos nuevos métodos constructores:*

- Uno con solo el *nombre* y la *familia* y el resto de los atributos numéricos a cero y si el atributo puedeSerDoméstico a false.
- Otro con el *nombre*, *familia* y *puedeSerDoméstico* y el resto de los atributos a cero.

Haga un test para construir dos objetos de tipo animal con los nuevos constructores y visualícelos.

¡¡Hala. A Eclipse 10 minutos para implementarlo y probarlo!!

Creación EXPRÉS de una clase

2023/24

Ejercicio de implementación "exprés" de un tipo.

Proyecto: *02_tipos* Paquete: *tipos* Clase *Electrodoméstico*

Propiedades:

- Número de Serie. String. Consultable.
- Descripción: String. Consultable.
- Meses de garantía: Integer. Consultable
- Importe: float. Consultable y modificable.
- Iva: float. Consultable y modificable.
- Importe Total: float. Consultable.

Métodos Constructores:

- Constructor_1: Debe tener un parámetro por cada propiedad básica.
- Constructor_2: Únicamente tendrá como parámetros el número de serie y la descripción. El resto de las propiedades básicas se inicializan a cero.

Métodos consultores y modificadores:

- Según se indica en la propiedad (*consultable y/o modificable*)

Creación EXPRÉS de una clase

2023/24

Ejercicio de implementación "exprés" de un tipo.

Proyecto: *02_tipos* Paquete: *tipos* Clase *Electrodoméstico*

Representación textual:

- Todas las propiedades.

Criterio de Igualdad:

- Por el número de serie y el importe

Criterio de ordenación

- Por el número de serie y a igualdad de número de serie desempata por el importe

Creación EXPRÉS de una clase

2023/24

Probando el tipo *Electrodoméstico*

Paquete: *tipos.test* Clase: *TestElectrodométicо*

Método *main*:

- Crear un objeto *e1* de tipo Electrodoméstico con el siguiente estado:
  "1111","Batidora",24,50,21
- Crear un objeto *e2* de tipo Electrodoméstico con el siguiente estado:
  "2222","Centro de Planchado",36,315.95,10
- Visualizar los dos objetos creados (*e1* y *e2*).
- Modificar el "iva" del segundo (*e2*) del 10 al 21%
- Visualizar la propiedad "Importe total" del segundo (*e2*)

2023/24

Implementación de restricciones

En la mayoría de los tipos que se implementan existen determinadas propiedades que no deberían tomar ciertos valores para que los objetos tengan sentido en el mundo real.

Por ejemplo:

- Una circunferencia no debe tener radio cero o negativo.
- Una persona no debe tener los apellidos y el nombre vacíos. O no puede nadie haber nacido en una fecha posterior al día de hoy.
- Un animal no debe tener pesoMedio o edadMedia negativo, o no puede tener el nombre en blanco.

Al hecho de controlar que los objetos no puedan tomar esos valores lo denominamos *implementación de restricciones*.

Implementación de restricciones

Implementación de restricciones

- Para controlar las restricciones, Java dispone de un mecanismo que se denomina lanzamientos de objetos excepciones (*throw*) que hace que el programa *detenga* su *ejecución*, informando al usuario por la consola de la restricción incumplida.

Sintaxis:

```
if (se incumple la restricción) {
        throw new nombre_del_tipo_excepción ("Texto informativo");
}
```

Implementación de restricciones

2023/24

Tipos Excepción más habituales:

En Java, podemos implementar nuestros propios tipos de excepciones, pero usaremos algunos ya implementados en JRE, por ejemplo:

- *IllegalArgumentException* (*normalmente usaremos esta*)
- *NullPointerException* (*y esta, también*)
- ArithmeticException
- IndexOutOfBoundsException
- NoSuchElementException
- …

Implementación de restricciones

¿Dónde se deben implementar las restricciones en una clase?

En aquellos métodos que asignen o modifiquen una propiedad con restricciones.

- Normalmente en los *constructores* y métodos *modificadores* que manejen la propiedad de que se trate.
- *NUNCA* en los métodos "gettters" o en otros que devuelvan propiedades derivadas.

Se trata de que un objeto no almacene un *ESTADO con un valor incorrecto* en alguna/s de sus propiedades. *Por lo que, en definitiva podemos decir que las restricciones se comprueban en los métodos que asignan valores a los atributos.*

2023/24

Implementación de restricciones

Ejemplo:

Restricciones del tipo Circunferencia:

- El radio debe ser positivo.

  En su caso, lanzar la excepción "*IllegalArgumentException*" con el mensaje "*El radio debe ser positivo*")

  (*recordar que Circunferencia tiene dos implementaciones y hay que hacer lo mismo en las dos*)

```
if (radio<=0) {
   throw new IllegalArgumentException
                ("El radio debe ser positivo");
}
```

Implementación de restricciones

2023/24

Ejemplo:

Probando las restricciones de Circunferencia:

Hacer un *testCircunferencia* en geometría.test en el que:

1. Se cree una circunferencia *c1* con radio 0, por ejemplo, con la implementación 2 y otra *c2* con radio positivo con la implementación 1. *En el fondo da igual las implementaciones que usemos*
2. Visualizar las dos circunferencias.
3. Se modifica el radio de *c2* a -20.
4. Visualizar otra vez las dos circunferencias.

Ejecutar el test y ver en la consola como salta la excepción.

Comentar la construcción de la primera circunferencia y *volver a ejecutar el test*

2023/24

Implementación de restricciones

¿Dónde se deben implementar las restricciones en los record?,

En caso de los *record* en el que sólo se asignan valores a los atributos en los constructores -hay que recordar que no hay "setters" ya que los objetos de un tipo record son inmutables-, se hace mediante un bloque "etiquetado".

Ejemplo en Persona:

```
public record Persona (String dni, String nombre,
       String apellidos, LocalDate fechaNacimiento) {
   …
       public Persona{
          if (fechaNacimiento.isAfter(LocalDate.now())) {
             throw new IllegalArgumentException("Error en la
                    fecha de nacimiento");
          }
       }
   …
}
```

2023/24

Implementación de restricciones

Implementación de restricciones

Ya se ha explicado que el lanzamiento de excepciones ***interrumpe*** el ***flujo*** del programa y el ***programa se aborta***, deteniendo su ejecución.

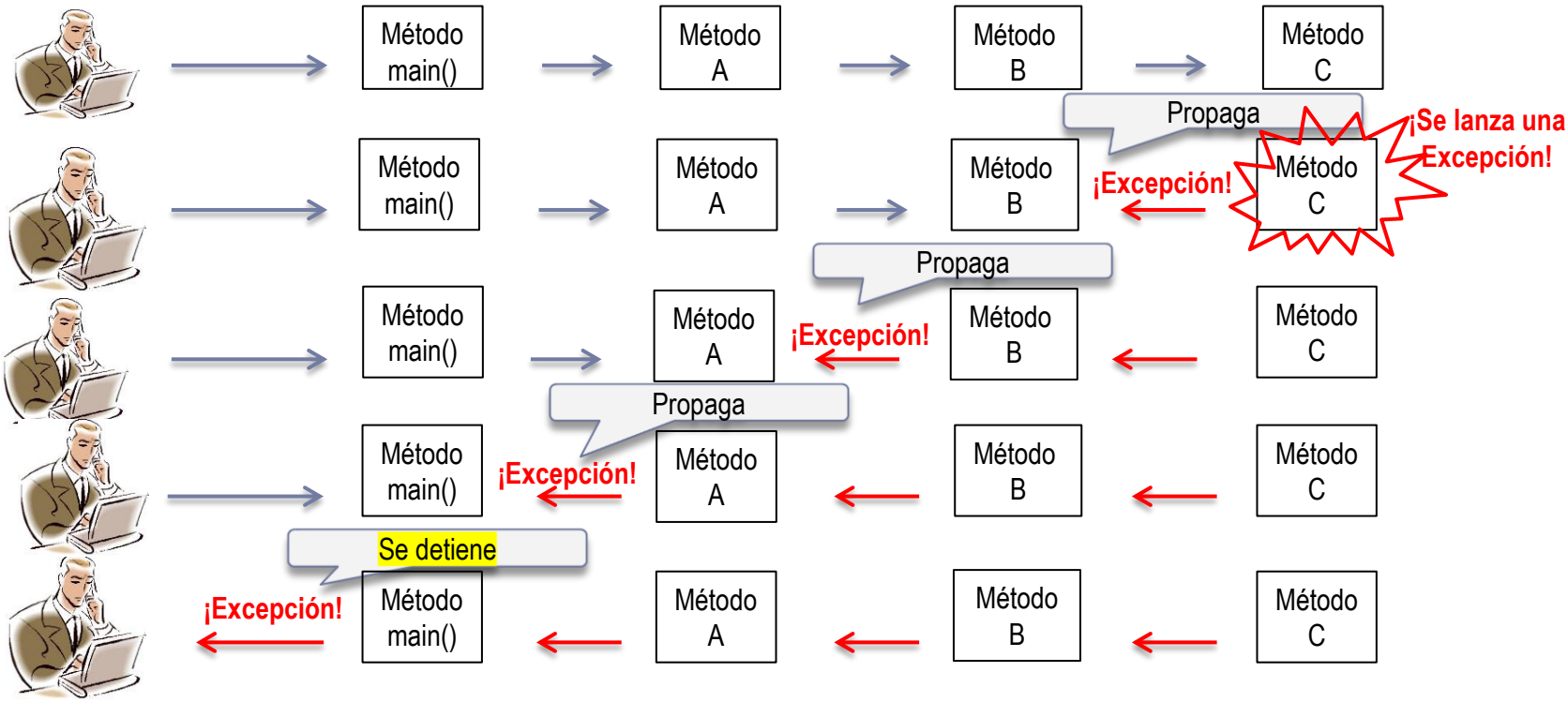
*En el caso de circunferencia hemos tenido que comentar la construcción de **c1** para llegar a la construcción de **c2***

Java también dispone de otro mecanismo para que gestionemos las excepciones ***sin que el programa se detenga***. Para esto último se usan los denominados bloques ***try-catch*** que permiten ***capturar*** las excepciones lanzadas. (*esto lo usaremos en los test*).

Propagación de Excepciones

Propagar la excepción:

El método ***aborta su ejecución*** y el programa se detiene. Por ejemplo:




```
java.lang.IllegalArgumentException: El radio debe ser positivo
    at geometría.CircunferenciaImpl2.<init>(CircunferenciaImpl2.java:9)
    at geometría.test.TestCircunferencia6.main(TestCircunferencia6.java:8)
```

2023/24

Captura de Excepciones try-catch

Cuando un método recibe una excepción que ha sido lanzada podemos evitar que se propague la excepción mediante su captura y gestionarla para que le programa no aborte:

Capturar la excepción: .

Es necesario incluir en el método que se quiere estudiar la excepción incluir un bloque *try-catch*. *Sólo es obligatorio un try con un bloque catch*

```
try{
    // Sentencias que incluyen llamadas a métodos
    // que pueden generar excepciones.
}catch(ExcepcionTipo1 e){
    // Sentencias de tratamiento de la excepción tipo 1.
}catch(ExcepcionTipo2 e){
    // Sentencias de tratamiento de la excepción tipo 2.
}finally{
   // Sentencias que se ejecutan siempre al final.
}
```

Captura de Excepciones try-catch

2023/24

Ejemplo:

Captura en el tipo Circunferencia:

Normalmente en nuestra asignatura las capturaremos en los test con el siguiente esquema:

```
public class TestCircunferencia7 {
  public static void main(String[] args) {
    testConstructor();
    testSetRadio();
  }
  private static void testConstructor() {
    Punto centro=new Punto(1.0,-1.0);
    System.out.println("\nTEST del Constructor correcto");
    try{
      Circunferencia c = new CircunferenciaImpl2(centro,3.0);
      System.out.println("c: "+ c);
    }catch(Exception e){
      System.out.println("Excepción capturada:\n " + e);
    }
```

Captura de Excepciones try-catch

2023/24

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
System.out.println("\nTEST del Constructor radio 0");
try{
     Circunferencia c = new CircunferenciaImpl2(centro,0d);
     System.out.println("c: "+ c);
}catch(Exception e){
   System.out.println("Excepción capturada:\n " + e);
}
System.out.println("\nTEST del Constructor radio 0");
try{
     Circunferencia c = new CircunferenciaImpl2(centro,0d);
     System.out.println("c: "+ c);
}catch(Exception e){
   System.out.println("Excepción capturada:\n " + e);
}
```

Captura de Excepciones try-catch

2023/24

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
private static void testSetRadio() {
   Punto centro=new Punto(1.0,-1.0);
   Circunferencia c = new CircunferenciaImpl1(centro,3.0);
   System.out.println("\nTEST setRadio con radio correcto");
   try {
      c.setRadio(10d);
      System.out.println("c: "+ c);
   }catch(Exception e) {
   System.out.println("Excepción capturada:\n " + e);
   }
   System.out.println("\nTEST setRadio con radio cero");
   try {
   c.setRadio(0d);
   System.out.println("c: "+ c);
   }catch(Exception e) {
   System.out.println("Excepción capturada:\n " + e);
   }
```

Captura de Excepciones try-catch

2023/24

Ejemplo:

Captura del tipo Circunferencia (continuación):

```
   private static void testSetRadio() {
      System.out.println("\nTEST setRadio con radio negativo");
      try {
         c.setRadio(-20d);
         System.out.println("c: "+ c);
      }catch(Exception e) {
      System.out.println("Excepción capturada:\n " + e);
      }
   }
}
```

2023/24

Implementación de restricciones

Hemos visto que para lanzar excepciones usamos una estructura

```
if (se incumple la restricción) {
    throw new nombre_del_tipo_excepción ("Texto informativo");
}
```

Si embargo, en la práctica, en nuestros proyectos usaremos la clase de utilidad *Checkers* con los métodos *ckeckNoNull* o *check*

Esta clase no forma parte de la API de Java sino que es una **gentileza** del departamento.

Implementación de restricciones

Sintaxis

- `Checkers.ckeckNoNull (parámetro1,parámetro2,parámetros3…);`
- `Checkers.ckeck ("Mensaje informativo", restricción a cumplir);`

Importante: en la estructura anterior, en la sentencia if se ponía la condición que provoca el error, en Checkers.check, el segundo parámetro contendrá la condición correcta

Ejemplo de lanzamiento de una excepción si el nombre o los apellidos de una persona son nulos:

```
Checkers.ckeckNoNull (nombre, apellidos);
```

Ejemplo de lanzamiento de una excepción si fa fecha de nacimiento es posterior al día de hoy :

```
Checkers.ckeck("la fecha de nacimiento no puede ser mayor de la actual",!fechaNacimiento.isBefore(LocalDate.now());
```