

Sesión 21: Excepciones

Programación 2

Ángel Herranz

Abril 2019

Universidad Politécnica de Madrid

En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos
- 👍 Tema 3: Programación Modular
- 👍 Tema 5: Herencia y Polimorfismo
- 🕒 Tema 6: Excepciones
- 🕒 Tema 7: Implementación de TADs lineales

Nodo<T>

En el capítulo de hoy



Tema 6: Excepciones

- Lo aplicamos a pilas acotadas



Tema 7: Implementación de TADs lineales

- Pilas no acotadas redimensionando arrays



Pilas acotadas

- API: crear, apilar, desapilar, cima, está vacía y está llena
- Vamos a aplicar mucho de lo que hemos aprendido
- PilaAcotada: interfaz genérico
- PilaAcotadaVacía: excepción para cima y desapilar
- PilaAcotadaLlena: excepción para apilar
- PilaArray: implementación de PilaAcotada
- TestPilaArray: tests para PilaAcotada

Introducir excepciones

1. **Analizar** el código de apoyo en las siguientes transparencias
2. **Elevar y declarar** las excepciones
3. **Capturar** las excepciones en el test

Código de apoyo

PilaAcotada

```
public interface PilaAcotada<T> {  
    void apilar(T dato) throws PilaAcotadaLlena;  
    T cima() throws PilaAcotadaVacía;  
    void desapilar() throws PilaAcotadaVacía;  
    boolean llena();  
    boolean vacía();  
}
```

PilaAcotadaLlena

```
public class PilaAcotadaLlena
    extends Exception
{
    public PilaAcotadaLlena() {
    }
}
```


PilaAcotadaVacía

```
public class PilaAcotadaVacía  
    extends Exception  
{  
    public PilaAcotadaVacía() {  
    }  
}
```

PilaArray i

```
public class PilaArray<T>
    implements PilaAcotada<T>
{
    private T[] datos;
    private int cima;

    @SuppressWarnings("unchecked")
    public PilaArray(int capacidad) {
        datos = (T[]) new Object[capacidad];
        cima = -1;
    }
}
```

PilaArray ii

```
public void apilar(T dato) throws PilaAcotadaLlena {  
    cima++;  
    datos[cima] = dato;  
}
```

```
public T cima() throws PilaAcotadaVacia {  
    return datos[cima];  
}
```

```
public void desapilar() throws PilaAcotadaVacia {  
    cima--;
```

PilaArray iii

```
}  
  
public boolean llena() {  
    return cima == datos.length - 1;  
}  
  
public boolean vacia() {  
    return cima == -1;  
}  
}
```

TestPilaArray i

```
public class TestPilaArray {  
    public static void main(String[] args) {  
        final int N = 10;  
        PilaAcotada<String> p = new PilaArray<String>(N);  
        String dato;  
        assert p.vacia();  
        assert !p.llena();  
        for (int i = 1; i < N; i++) {  
            dato = "Dato-" + i;  
            p.apilar(dato);  
            assert !p.vacia();  
        }  
    }  
}
```

TestPilaArray ii

```
    assert !p.llena();
    assert p.cima().equals(dato);
}
dato = "Dato-"+N;
p.apilar(dato);
assert p.llena();
for (int i = N; i > 1; i--) {
    dato = "Dato-"+i;
    assert !p.vacia();
    assert p.cima().equals(dato);
    p.desapilar();
    assert !p.llena();
}
```

TestPilaArray iii

```
    }  
    dato = "Dato-"+1;  
    assert !p.vacia();  
    assert !p.llena();  
    assert p.cima().equals(dato);  
    p.desapilar();  
    assert p.vacia();  
    assert !p.llena();  
}  
}
```