



ORACLE

Academy



Java Foundations

7-1

Creación de una clase

ORACLE
Academy



Objetivos

- En esta lección se abordan los siguientes objetivos:
 - Crear una clase principal/de prueba Java
 - Cree una clase Java en el IDE
 - Usar condicionales en métodos
 - Traducir las especificaciones o una descripción en campos y comportamientos



Conceptos orientados a objetos

- Hemos estado experimentando con sentencias condicionales y bucles durante un tiempo
- Ahora es un buen momento para revisar los conceptos de la programación orientada a objetos y sus ventajas
- En el resto de esta sección se describe la programación orientada a objetos con más detalle

Ejercicio 1



- Juegue a los rompecabezas básicos 6 y 7
 - <https://objectstorage.uk-london-1.oraclecloud.com/n/lrvrlgaqj8dd/b/Games/o/JavaPuzzleBall/index.html>
 - Su objetivo: Diseñar una solución que desvíe la pelota a Duke
- Tenga en cuenta lo siguiente:
 - ¿Qué ocurre cuando se coloca un icono en la rueda azul?





Análisis de Java Puzzle Ball

- ¿Qué ocurre cuando coloca iconos dentro de una rueda azul?
 - Aparece una pared en cada instancia de un objeto deflector azul
 - Las paredes proporcionan comportamientos que desvían e interactúan con la bola
 - Todas las instancias de los deflectores azules comparten estos mismos comportamientos

WebCenter
Sites



WebCenter
Sites

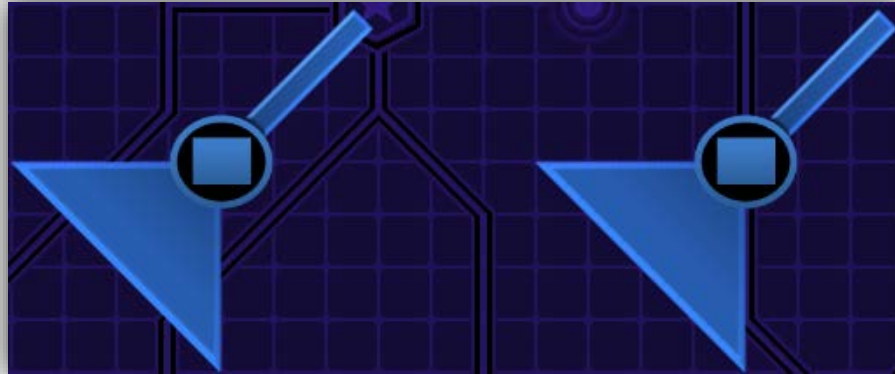


Descripción de un deflector azul

- Propiedades:

- Color
- Forma
- Posición x
- Posición y

(Campos)



- Comportamientos:

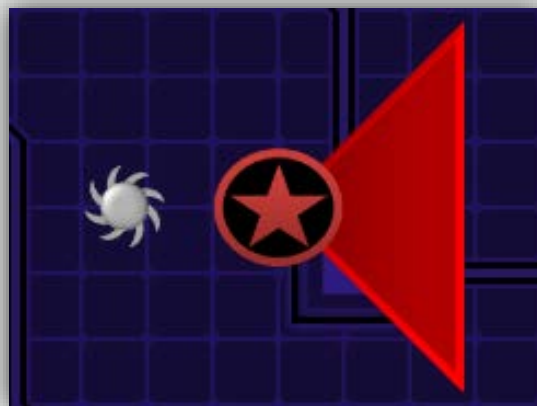
- Hacer sonido de ping
- Parpadear
- Desviar bola
- Destruirse

(Métodos)



Lógica condicional y bucles en clases

- Las condiciones y los bucles pueden desempeñar un papel importante en los métodos que escriba para una clase
- El método main era un lugar adecuado para experimentar y aprender la lógica condicional y los bucles
- Pero recuerde...
 - El método main está diseñado para ser una clase de controlador
 - En el método main no debe estar escrito todo el programa

¿Qué ocurre si la bola colisiona contra un deflector?



- Se llama a un método con la siguiente lógica:

```
public void onCollisionWithBall(Ball ball){  
    if(ball.isBlade == true){           //Ball is blade   
        getDestroyed();  
    }  
    else{                               //Ball is not blade   
        deflectBall();  
    }  
}
```

Modelación de una cuenta de ahorro

- Podría modelar una cuenta de ahorro de este modo:

```
public class SavingsAccount{  
  
    public static void main(String args[]){  
  
        int balance = 1000;  
        String name = "Damien";  
    }//end method main  
}//end class SavingsAccount
```

- Y dos cuentas como se muestra a continuación:

```
int balance1 = 1000;  
String name1 = "Damien";  
  
int balance2 = 2000;  
String name2 = "Bill";           //Copy, Paste, Rename
```

Modelación de muchas cuentas

- ¿Cómo modelaría 1000 cuentas?

```
...  
//You think ...  
//Do I really have to copy and paste 1000 times?
```

- ¿Cómo agregaría un parámetro para cada cuenta?

```
...  
//You think ...  
//There has to be a better way!
```

- Hay una mejor forma:
 - utilizar una clase
 - Y no el método main

Cómo estructurar una clase

- El código se debe ajustar a este formato:

```
1 public class SavingsAccount {  
2  
3     Properties  
4  
5  
6     Behaviors  
7  
8  
9 }
```

Cómo estructurar una clase

- El código se debe ajustar a este formato:

```
1 public class SavingsAccount {  
2     public double balance;  
3     public double interestRate = 0.01;  
4     public String name;  
5  
6     public void displayCustomer(){  
7         System.out.println("Customer: "+ name);  
8     }//end method displayCustomer  
9 }//end class SavingsAccount
```

- Con una sola línea de código (línea 3), las 1000 cuentas tienen un tipo de interés
 - Y podemos cambiar el tipo en cualquier momento para cualquier cuenta

El método main como una clase de controlador

- Incluya el método main una clase de prueba
 - El método main se suele usar para la instanciación

```
public class AccountTest {  
    public static void main(String[] args){  
  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.name = "Damien";  
        sa0001.interestRate = 0.02;  
  
        SavingsAccount sa0002 = new SavingsAccount();  
        sa0002.balance = 2000;  
        sa0002.name = "Bill";  
    } //end method main  
} //end class AccountTest
```


Ejercicio 2

- Cree un nuevo proyecto Java
- Cree una clase `AccountTest` con un método `main`
- Cree una clase `CheckingAccount`
 - Incluya los campos `balance` y `name`
- Instancie un objeto `CheckingAccount` desde el método `main`
 - Asigne valores a los campos `balance` y `name` del objeto

Ámbito de las variables

- Se puede acceder a los campos en cualquier parte de una clase
 - Esto incluye los métodos

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    public void displayCustomer(){  
        System.out.println("Customer: " + name);  
        System.out.println("Balance: " + balance);  
        System.out.println("Rate: " + interestRate);  
    } //end method displayCustomer  
} //end class SavingsAccount
```

Ámbito de las variables

- No se puede acceder a las variables creadas en un método desde fuera de él
 - Esto incluye los parámetros de métodos

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;
```

```
    public void deposit(int x){  
        balance += x;  
    } //end method deposit
```

Ámbito de x

```
    public void badMethod(){  
        System.out.println(x);  
    } //end method badMethod  
} //end class SavingsAccount
```

No es ámbito de x

Acceso a campos y métodos de otra clase

1. Cree una instancia
2. Utilice el operador de punto (.)

```
public class AccountTest {  
    public static void main(String[] args){  
        1) SavingsAccount sa0001 = new SavingsAccount();  
        2) { sa0001.name = "Damien";  
            sa0001.deposit(1000);  
        }  
    }  
}
```


```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    }  
}
```

Transferencia de valores a métodos

- 1000 se transfiere al método deposit()
- El valor de x se convierte en 1000

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.name = "Damien";  
        sa0001.deposit(1000);  
    }//end class AccountTest
```

```
public class SavingsAccount {  
    public String name;  
    public double balance;  
  
    public void deposit(int x){  
        balance += x;  
    }//end method deposit  
}//end class SavingsAccount
```



x = 1000

Ejercicio 3

- Continúe la edición del proyecto `AccountTest`
- Escriba un método `withdraw()` para cuentas corrientes que...
 - Acepta un argumento `double` para la cantidad que se deba retirar
 - Imprime una advertencia si el saldo es demasiado bajo para realizar la retirada
 - Imprime una advertencia si el argumento de retirada es negativo
 - Si no hay advertencias, el importe de retirada se resta del saldo. Imprima el nuevo saldo
- Pruebe este método con la instancia del ejercicio 2

¿Qué ocurre si necesito un valor de un método?

- Las variables están restringidas por su ámbito
- Pero es posible obtener el valor de estas variables desde fuera de un método

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;
```

```
    public void calcInterest(){  
        double interest = balance*interestRate/12;  
  
    }//end method calcInterest
```

Ámbito de
interés

```
}//end class SavingsAccount
```

Devolución de valores desde métodos

- Si desea obtener un valor de un método...
 - Escriba una sentencia return
 - Cambiar el tipo de método de void al tipo que desee devolver

```
public class SavingsAccount {  
    public double balance;  
    public double interestRate;  
    public String name;  
  
    //This method has a double return type  
    public double calcInterest(){  
        double interest = balance * interestRate / 12;  
        return interest;  
    } //end method calcInterest  
} //end class SavingsAccount
```

Devolución de valores: ejemplo

- Cuando `getInterest()` devuelve un valor...

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += sa0001.calcInterest();  
    } //end class AccountTest
```

- Equivale a escribir...

```
public class AccountTest {  
    public static void main(String[] args){  
        SavingsAccount sa0001 = new SavingsAccount();  
        sa0001.balance = 1000;  
        sa0001.balance += 0.83;  
    } //end class AccountTest
```

- Pero es mejor y más flexible porque el valor se calcula en lugar de codificarse

Resumen de los métodos

The diagram illustrates the components of a Java method signature using the example `public double calculate(int x, double y){`. Red brackets are used to group parts of the signature, with labels above them: 'Tipo de devolución de método' points to 'public double', 'Nombre del método' points to 'calculate', and 'Parámetros' points to '(int x, double y)'. A red bracket on the right side of the code block groups the implementation lines with the label 'Implantación'.

```
public double calculate(int x, double y){  
    double quotient = x/y;  
    return quotient;  
}//end method calculate
```

Limitación del método main

- El método main debe ser tan pequeño como sea posible
- El siguiente ejemplo no es muy bueno porque...
 - Aumentar el saldo de una cuenta según el interés es un comportamiento típico de las cuentas
 - El código de este comportamiento se debe escribir como un método en la clase SavingsAccount
 - También es peligroso tener un programa de cuentas donde el campo de saldo se pueda manipular libremente

```
public static void main(String[] args){  
    SavingsAccount sa0001 = new SavingsAccount();  
    sa0001.balance = 1000;  
    sa0001.balance += sa0001.calcInterest();  
}//end method main
```

Resto de la sección

- Aprenderemos a evitar estos escenarios problemáticos al desarrollar una clase
- Pero en esta lección, solo nos centraremos en conocer cómo:
 - Interpretar una descripción o especificación
 - Dividirla en propiedades y comportamientos
 - Traducir esas propiedades y comportamientos en campos y métodos

Ejercicio 4

- Continúe la edición del proyecto `AccountTest`
- Cree una nueva clase según la descripción
- Asegúrese de instanciar esta clase y de probar sus métodos
 - Cree un bono de ahorro
 - Una persona puede comprar un bono para cualquier plazo de entre 1 y 60 meses
 - Un bono obtiene un interés cada mes hasta que vence su plazo (0 meses restantes)
 - El plazo y el tipo de interés se definen al mismo tiempo
 - El tipo de interés del bono se basa en el plazo según el siguiente siempre de niveles:

0–11 meses	: 0,5%
12–23 meses	: 1,0%
24–35 meses	: 1,5%
36–47 meses	: 2,0%
48–60 meses	: 2,5%

Descripción de un bono de ahorro

- Propiedades:

- Nombre
- Saldo
- Plazo
- Meses restantes
- Tipo de interés



- Comportamientos:

- Definir el tipo de interés según el plazo
- Obtener interés
- Vencimiento (0 meses restantes)



Conversión a código Java: parte 1

- La clase Bond puede representado los campos del siguiente modo:

```
public class Bond{  
    public String name;  
    public double balance, rate;  
    public int term, monthsRemaining;
```

El código continúa en la siguiente diapositiva...



Conversión a código Java: parte 2

- Incluya los siguientes métodos:

```
public void setTermAndRate(int t){  
    if(t>=0 && t<12)  
        rate = 0.005;  
    else if(t>=12 && t<24)  
        rate = 0.010;  
    else if(t>=24 && t<36)  
        rate = 0.015;  
    else if(t>=36 && t<48)  
        rate = 0.020;  
    else if(t>=48 && t<=60)  
        rate = 0.025;  
    else{  
        System.out.println("Invalid Term");  
        t = 0;  
    }  
    term = t;  
    monthsRemaining = t;  
} //end method setTermAndRate
```

El código continúa en la siguiente diapositiva...



Conversión a código Java: parte 3

```
public void earnInterest(){
    if(monthsRemaining > 0){
        balance += balance * rate / 12;
        monthsRemaining--;
        System.out.println("Balance: $" +balance);
        System.out.println("Rate: " +rate);
        System.out.println("Months Remaining: "
                            + monthsRemaining);
    }
    else{
        System.out.println("Bond Matured");
    }//endif
} //end method earnInterest
} //end class Bond
```

Resumen

- En esta lección, debe haber aprendido lo siguiente:
 - Crear una clase principal/de prueba Java
 - Cree una clase Java en el IDE
 - Usar condicionales en métodos
 - Traducir las especificaciones o una descripción en campos y comportamientos





ORACLE

Academy

