

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

# ORACLE

## Academy

# Java Foundations

8-4

Técnicas y conceptos de depuración

**ORACLE**  
Academy



Copyright © 2022, Oracle y/o sus filiales. Oracle, Java y MySQL son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

# Objetivos

- En esta lección se abordan los siguientes objetivos:
  - Probar y depurar un programa Java
  - Identificar los tres tipos de errores
  - Aplicar técnicas de depuración
    - Sentencias `print`
    - Uso del depurador de su IDE
  - Aplicar algunos consejos y técnicas de depuración



# Prueba de un programa Java

- Richie ha escrito un programa Java para encontrar el máximo de tres enteros:

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > num2 && num1 > num3) {  
        max = num1;  
    }//endif  
    if (num2 > num1 && num2 > num3) {  
        max = num2;  
    }//endif  
    if (num3 > num1 && num3 > num2) {  
        max = num3;  
    }//endif  
    System.out.println("The max of 3 numbers is " + max);  
}//end method main
```

## Prueba de un programa Java

- Richie lo ha probado en muchos juegos de datos, como <3,5,9>, <12,1,6> y <2,7,4>
- El programa funciona para todos los datos
- No obstante, se le ha dicho que el programa no funciona y no sabe el motivo

## Ejercicio 1

- Cree un nuevo proyecto y agréguele el archivo `MaxIntegers.java`
- Observe `MaxIntegers.java`
  - ¿Puede identificar qué es lo que no ha incluido Richie en su prueba?

## Identificar el error

- El programa falla cuando se prueba con valores duplicados, como  $\langle 3,3,3 \rangle$  y  $\langle 7,2,7 \rangle$ , y muestra la salida como cero
  - Ha identificado el error
  - El siguiente paso consiste en corregir el error

## Corregir el error

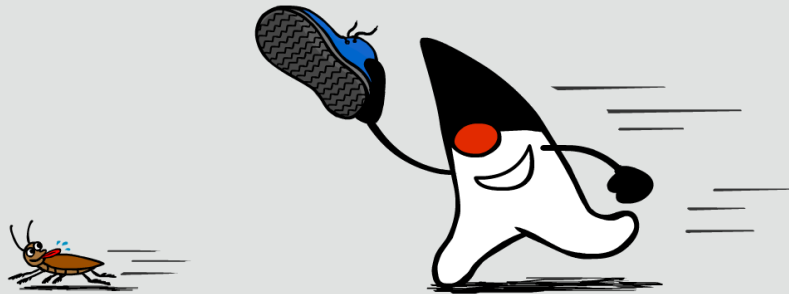
- Modifique el programa y pruébelo en muchos juegos de datos, incluidos los valores duplicados

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > max) {  
        max = num1;  
    }//endif  
    if (num2 > max) {  
        max = num2;  
    }//endif  
    if (num3 > max) {  
        max = num3;  
    }//endif  
    System.out.println("The max of 3 numbers is " + max);  
}//end method main
```



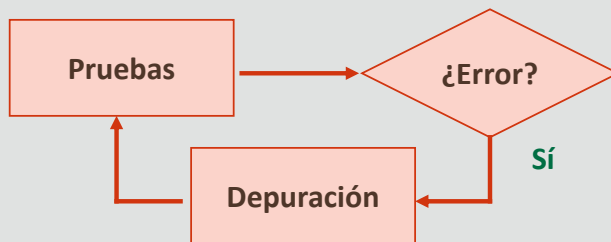
# Prueba y Depuración

- Como ha observado en el ejemplo anterior, la prueba y la depuración son actividades importantes en el desarrollo de software



# Prueba y Depuración

- Prueba:
  - Determinar si un código contiene errores
- Depuración:
  - Identificar un error y corregirlo



# Tres tipos de errores

- Errores
  - Errores de compilación
  - Errores de lógica
  - Errores de tiempo de ejecución

# Errores de compilación

- Error de sintaxis
- Tipo de errores más sencillo de corregir
- Ejemplos:

- Ejemplo 1: Falta el punto y coma

- `int a = 5 // falta el punto y coma`

- Ejemplo 2: Errores en la expresión

- `x = ( 3 + 5; //falta el paréntesis de //cierre`
    - `y = 3 + * 5; //falta el argumento entre // '+' y '*'`

## Errores de lógica

- El programa se ejecuta pero genera un resultado incorrecto
- Es difícil de detectar, por lo que es más difícil de corregir
- Ejemplo: Variable no inicializada

```
- int i;  
- i++; // la variable i no está inicializada
```

# Errores de tiempo de ejecución

- Estos errores se producen en tiempo de ejecución
- El mecanismo de manejo de excepciones de Java puede detectar estos errores
- Algunas de las excepciones habituales:
  - `ArrayIndexOutOfBoundsException`
  - `NullPointerException`
  - `ArithmeticException`

# Técnicas de depuración

- Veamos dos técnicas de depuración:
  - Uso de sentencias `print`
  - Uso de un depurador IDE

# Sentencias `print`: Ventajas

- Fáciles de agregar
- Proporcionan información
  - Métodos a los que se ha llamado
  - Valor de los parámetros
  - Orden en el que se ha llamado a los métodos
  - Valores de las variables y los campos locales en puntos estratégicos



## Sentencias `print`: Desventajas

- No resulta práctico agregar sentencias `print` a cada método
- Si hay demasiadas sentencias `print`, se produce una sobrecarga de información
- La eliminación de las sentencias `print` es tediosa

## Sentencias print: Ejemplo

- Tenga en cuenta este código Java:

```
int n = 10;
int sum = 10;
while (n > 1){
    sum = sum + n;
    n--;
} //end while
System.out.println("The sum of the integers 1 to 10 is " + sum);
```

- Al ejecutar el programa, no funciona correctamente
- Para averiguar cuál es el problema, puede rastrear el valor de las variables n y sum insertando sentencias print

## Programa modificado con sentencias `print` adicionales para la depuración

```
int n = 10;
int sum = 10;
while (n > 1) {
    System.out.println("At the beginning of the loop: n = " + n);
    System.out.println("At the beginning of the loop:sum=" + sum);
    n--;
    System.out.println("At the end of the loop: n = " + n);
    System.out.println("At the end of the loop: sum = " + sum);
}
System.out.println("The sum of the integers 1 to 10 is " + sum);
```

# Salida

- A continuación se muestran las primeras cuatro líneas de la salida después de la primera iteración del bucle:
  - At the beginning of the loop:  $n = 10$
  - At the beginning of the loop:  $\text{sum} = 10$
  - At the end of the loop:  $n = 9$
  - At the end of the loop:  $\text{sum} = 20$
- Puede ver que hay algo incorrecto:
  - La variable `sum` se ha definido en 20
  - Debido a que se ha inicializado en 10, se define en  $10 + 10$ , que resulta incorrecto si desea sumar los números del 1 al 10

## Depurador IDE

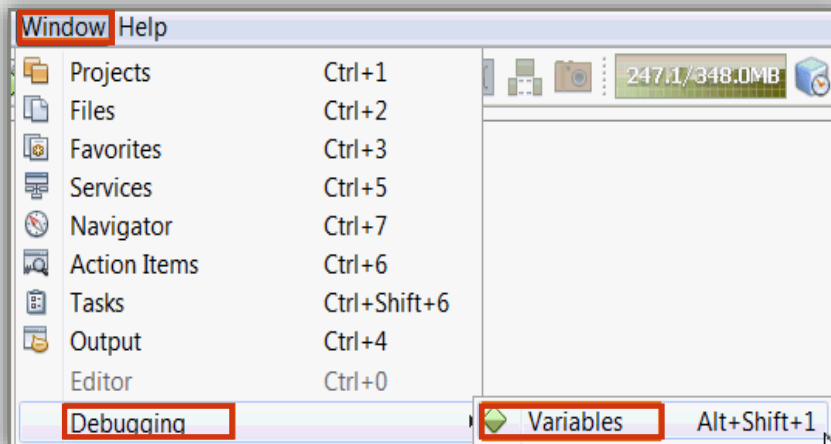
- Ya ha utilizado el entorno de depuración gráfico IDE
- Ha utilizado las siguientes funciones del depurador:
  - Definir puntos de ruptura
  - Rastrear un programa línea a línea
- Vamos a usar otra función muy importante para ver el contenido de las variables

## Ventana Variables

- Cuando llega a un punto de ruptura definido, puede usar la ventana Variables para ver el valor de las variables en ese momento
- Puede averiguar los valores de las variables sin tener que poner muchas sentencias print en el programa
- En las siguientes diapositivas se muestra el uso del depurador en NetBeans
- Si está utilizando otro IDE, consulte la documentación para obtener información sobre cómo realizar este proceso

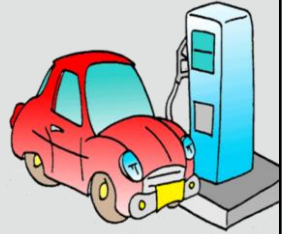
## Acceso a la ventana Variables

- Para ver la ventana Variable, en el menú principal de NetBeans:
  - Haga clic en Window > Debugging > Variables




## Ejercicio 2: Escenario

- Supongamos que tiene un coche y que quiere ir a la gasolinera Tiene la siguiente información:
  - Posición actual del coche:  $x_1$  e  $y_1$
  - Ubicación de la gasolinera:  $x_2$  e  $y_2$
  - Velocidad del coche
- Desea calcular el tiempo que tardará el coche desde su posición actual ( $x_1, y_1$ ) hasta llegar a la gasolinera ( $x_2, y_2$ )
- En el proyecto ComputeTime hay disponible un programa Java para calcular el tiempo con la fórmula  $\text{tiempo} = \text{distancia} / \text{velocidad}$



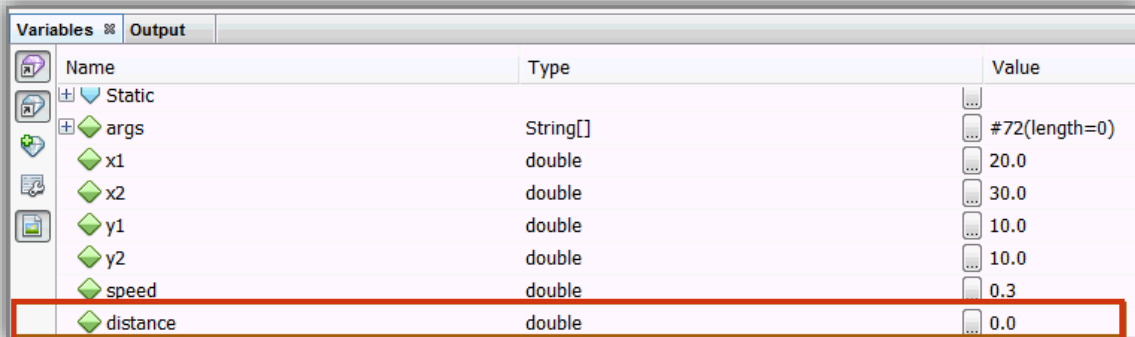


## Ejercicio 2

- Agregue el archivo `ComputeTime.java` al proyecto creado para el ejercicio 1
- Observe `ComputeTime.java`
- Ejecute el programa con el depurador del IDE para depurar este programa:
  - Defina el punto de ruptura en el método `getDistance`
  - Haga clic en Step In para ir a la línea siguiente 
  - Observe los valores de las variables `x1`, `x2`, `y1`, `y2`, `speed`, `distance` y `time`
- ¿Puede identificar el error?

## Observar el valor de distance

- En el ejercicio anterior utilizó las funciones de depuración de su IDE para identificar el error:



Name	Type	Value
Static		
args	String[]	#72(length=0)
x1	double	20.0
x2	double	30.0
y1	double	10.0
y2	double	10.0
speed	double	0.3
distance	double	0.0

- Como puede ver, el valor de distance es 0.0, la fórmula para calcular la distancia era errónea y ha provocado un valor de devolución incorrecto para la variable distance

# Identificación del error potencial

```
public static void main(String[] args) {  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance/speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
  
} //end method main  
  
static double getDistance(double x1, double x2, double y1, double y2){  
    return Math.sqrt((x1 - x1) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
} //end method getDistance
```

Error potencial

## Corrección del error

- Como ha identificado el error, puede cambiar la ubicación del punto de ruptura a la posición donde se llama al método `getDistance()`
- De este modo se ahorra tener que recorrer el código que ya ha examinado
- Por lo tanto, modifique el código y vuelva a ejecutar el depurador con el nuevo punto de ruptura para ver qué obtenemos

# Nueva ejecución del depurador

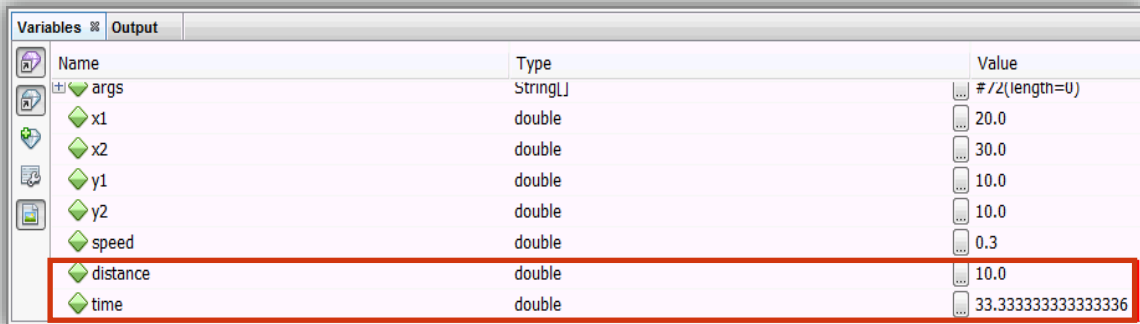
```
public static void main(String[] args) {  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance/speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
  
} //end method main  
  
static double getDistance(double x1, double x2, double y1, double y2){  
    return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
} //end method getDistance
```

Nuevo punto de ruptura

Código modificado

# Observación de las variables

- Hemos corregido el error
  - La variable distance ahora indica el valor 10.0 y la variable time ahora indica el valor 33.33



Name	Type	Value
args	String[]	#/2(length=0)
x1	double	20.0
x2	double	30.0
y1	double	10.0
y2	double	10.0
speed	double	0.3
distance	double	10.0
time	double	33.333333333333336

## Operador de un solo signo igual frente a dos signos igual

- Operador de asignación (=) frente a comparación (==)
  - 1. Operador de comparación
    - `if(x = 0)` en vez de `if(x == 0)`
    - Se encuentra en las sentencias `if`, `for` y `while`.
  - 2. Operador de asignación
    - `int x == 1;` en vez de `int x = 1;`

## Punto y coma en un lugar erróneo

- Busque el punto y coma después de la sentencia if o las sentencias de bucle for/while

```
if (x == 0); {  
    <statements>  
}
```

instead of

```
if(x == 0) {  
    <statements>  
}
```

```
while(<boolean expression>;) {  
    <statements>  
}
```

instead of

```
while(<boolean expression>) {  
    <statements>  
}
```



# Llamada a métodos con argumentos erróneos

- Los tipos de parámetro de llamada a método deben coincidir con los tipos de parámetro de definición de método
- Por ejemplo:
  - Dada una definición de método:
    - `void methodName(int x, char y) { }`
  - Llame a este método:
    - `methodName(a, b)`



a debe ser del tipo int y b, del tipo char

## Condiciones de límite

- Es importante probar las condiciones de límite
- La lógica subyacente a las pruebas que se hagan de ellas es que los errores suelen producirse cerca de los valores de límite de una variable de entrada
- Por ejemplo, una condición de límite para:
  - Datos de entrada (prueba de válido frente a no válido)
  - Bucles (principio y final de bucles)

## Prueba de las condiciones de límite de los bucles

- Permite realizar pruebas de situaciones de límite como “menor que” y “mayor que” para que las condiciones de iteración de bucle se prueben de forma precisa
- Por ejemplo, este bucle:

```
if ( num >= 50 && num <= 100 ) {  
    //do stuff  
} //endif
```

- Para probar las condiciones de límite, se probarían números cerca de 50 y 100, es decir, 49, 50, 51, 99, 100 y 101

## Ejercicio 3

- Agregue el archivo `BoundaryTesting.java` al proyecto creado para el ejercicio 1
- Observe `BoundaryTesting.java`
- Valide la entrada ejecutando el programa con los siguientes valores de prueba de límite para el año y el mes:

Año	Mes
1582	2
1583	0
1583	13
1583	1
1583	12

# Resumen

- En esta lección, debe haber aprendido lo siguiente:
  - Probar y depurar un programa Java
  - Identificar los tres tipos de errores
  - Aplicar técnicas de depuración
    - Sentencias `print`
    - Uso del depurador de su IDE
  - Aplicar algunos consejos y técnicas de depuración



**ORACLE**  
Academy

JFo 8-4  
Técnicas y conceptos de depuración

Copyright © 2022, Oracle y/o sus filiales. Oracle, Java y MySQL son marcas comerciales registradas de Oracle y/o sus filiales. Todos los demás nombres pueden ser marcas comerciales de sus respectivos propietarios.

37

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

# ORACLE

## Academy