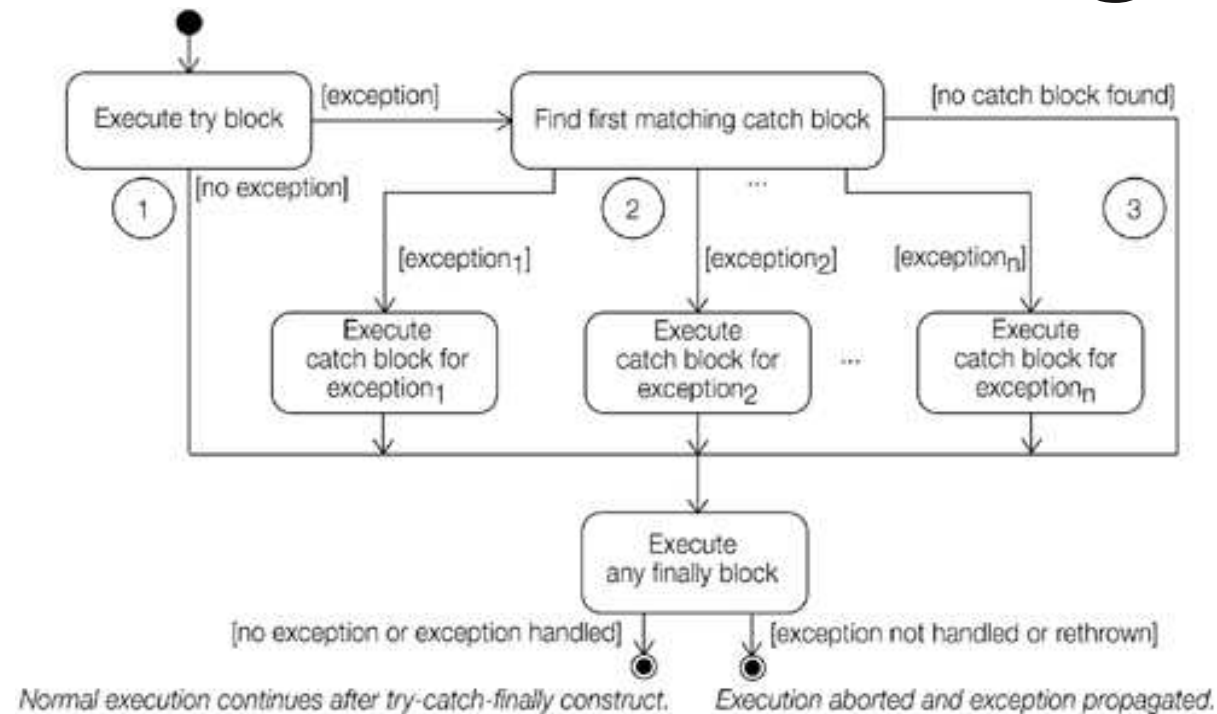


# Session 09: Exception Handling



# Table of contents

**1 Java Exception**

**2 Exception Handling**

**3 Checked And Unchecked Exception**

**4 Throw and Throws**

# Java Exceptions (1)

## Introduction Exception

- During the execution of a program, the computer will face the some of situations:
  - **Syntax error**
  - **Logic algorithm error**
  - **Runtime error**
- An exception is an **abnormal condition** that arises[xuất hiện] in a code sequence at **run time**

# Java Exceptions (2)

## Introduction Exception

- The exception handling is one of the powerful mechanism to **handle the runtime errors** so that **normal flow of the application** can be maintained

- Example

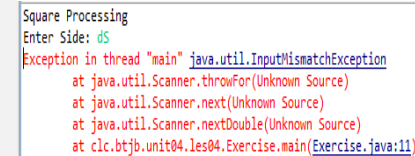
```
statement 1;  
statement 2;  
statement 3;  
statement 4;  
statement 5; //exception occurs  
statement 6;  
statement 7;
```

# Java Exceptions (3)

## Introduction Exception

- Example: A program that requests a number from the user

```
public class Exercise {  
    public static void main(String[] args) {  
        double side;  
        Scanner s = new Scanner(System.in);  
  
        System.out.print("Enter Side: ");  
        side = s.nextDouble();  
  
        System.out.printf("Perimeter: %.2f\n", side * 4);  
    }  
}
```



```
Square Processing  
Enter Side: dS  
Exception in thread "main" java.util.InputMismatchException  
    at java.util.Scanner.throwFor(Unknown Source)  
    at java.util.Scanner.next(Unknown Source)  
    at java.util.Scanner.nextDouble(Unknown Source)  
    at clc.btb.unit04.Les04.Exercise.main(Exercise.java:11)
```

# Java Exceptions (4)

## Introduction Exception

- Runtime errors such as:
  - reference of a **null pointer**
  - **out-of-bounds** array access
  - **divide by zero**
  - attempt to open a **non-existent file** for reading
  - **bad cast** (e.g., casting an Object that is actually a Boolean to Integer)

# Java Exceptions (5)

## Introduction Exception

- There are mainly two types of exceptions
  - **checked**
  - **unchecked**
- Here, an error is considered as the unchecked exception
- Exception Keywords
  - **try**
  - **catch**
  - **finally**
  - **throw/throws**

2

# Exception Handling



# Exception Handling (1)

## Introduction

- When an **exceptional condition** arises, an **object** representing that exception is **created** and **thrown in the method** that caused the error
- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**
  - Program statements that you want to monitor for exceptions are contained within a **try** block
  - Your code can catch this exception (using **catch**) and handle it in some rational manner

# Exception Handling (2)

## Syntax

- **try-catch** block

```
try {  
    // code that may throw an exception  
} catch (Exception_Class_Name ref) {  
    // your code can catch this exception  
}
```

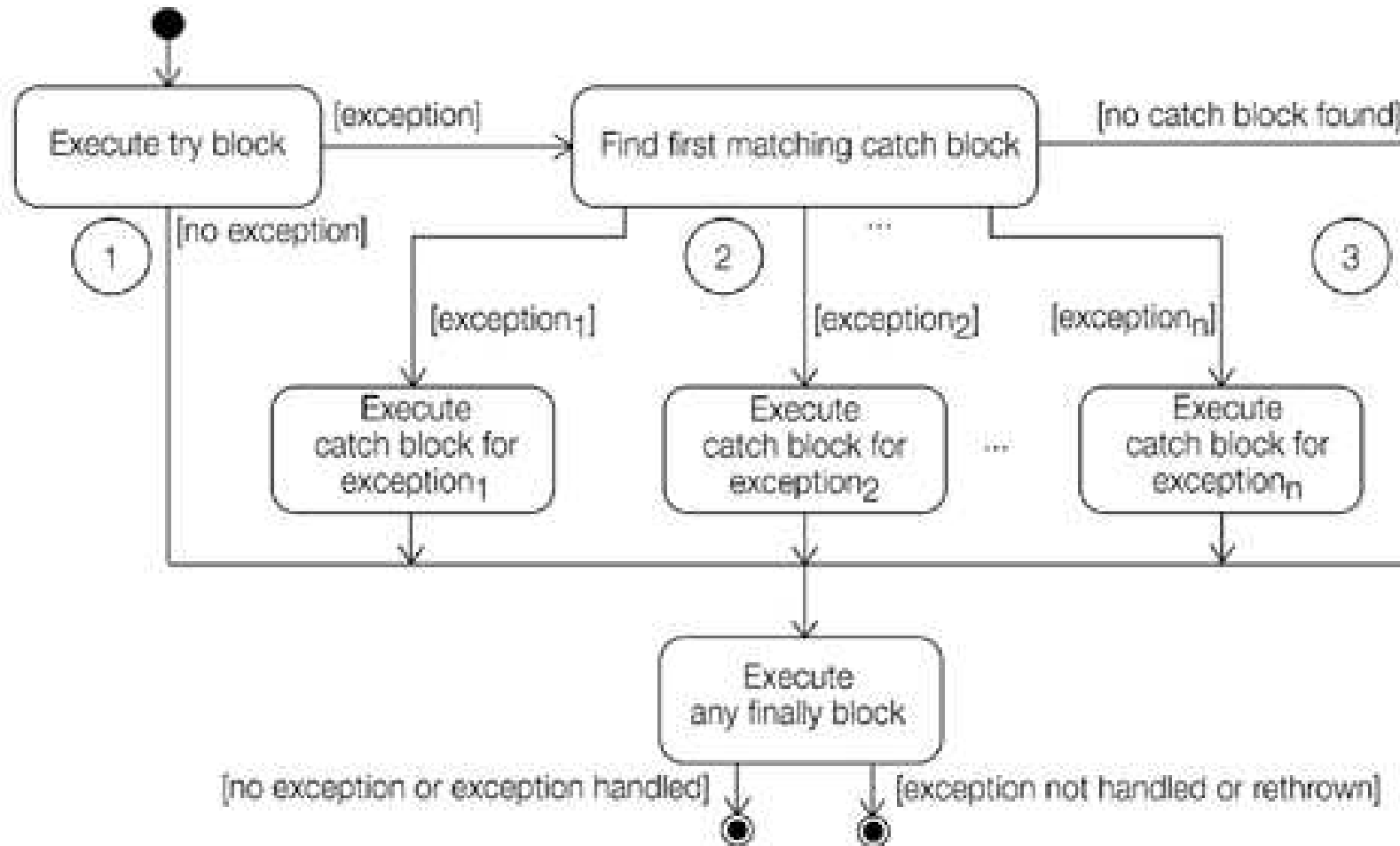
- **try-finally & try-catch-finally** block

```
try {  
    // code that may throw an exception  
} finally {  
    // clean up  
}
```

```
try {  
    // code that may throw an exception  
} catch (Exception_Class_Name ref) {  
    // your code can catch this exception  
} finally {  
    // clean up  
}
```

# Exception Handling (3)

## Flow chart



*Normal execution continues after try-catch-finally construct.*

*Execution aborted and exception propagated.*

# Exception Handling (4)

## Example

```
try {  
    Scanner s = new Scanner ("System.in");  
    side = s.nextDouble();  
  
    System.out.println("Square Characteristics");  
    System.out.printf("Side: %.2f\n", side);  
  
    System.out.printf("Perimeter: %.2f\n", side * 4);  
} catch(InputMismatchException ex) {  
    System.err.println(ex.getMessage());  
}
```

# Exception Handling (5)

## Multi-catch block

- A try block can be followed by **one or more** catch blocks. Each catch block must contain a **different** exception handler
- You have to perform **different tasks** at the occurrence of different exceptions, use java multi-catch block

```
try {  
    // block of code to monitor for errors  
} catch (ExceptionType1 exOb) {  
    // exception handler for ExceptionType1  
} catch (ExceptionType2 exOb) {  
    // exception handler for ExceptionType2  
}
```

# Exception Handling (6)

## Multi-catch block

- At a time only one exception occurs and at a time **only one catch block is executed**
- All catch blocks **must be ordered from most specific to most general**, i.e. catch for `ArithmeticException` must come before catch for `Exception`

```
try {  
  
} catch (ArithmeticException ex1) {  
    System.err.println(ex1.getStackTrace());  
} catch (ArrayIndexOutOfBoundsException ex2) {  
    System.err.println(ex2.getStackTrace());  
} catch (FileNotFoundException ex3) {  
    System.err.println(ex3.getStackTrace());  
}
```

# Exception Handling (7)

## Multi-catch block

```
public class MultipleCatchBlock {  
    public static void main(String[] args) {  
        try {  
            int a[] = new int[5];  
            a[5] = 30 / 0;  
            System.out.println(a[10]);  
        } catch (ArithmeticException e) {  
            System.err.println("Arithmetic Exception occurs");  
        } catch (ArrayIndexOutOfBoundsException e) {  
            System.err.println("ArrayIndexOutOfBoundsException occurs");  
        } catch (Exception e) {  
            System.err.println("Parent Exception occurs");  
        }  
        System.out.println("rest of the code");  
    }  
}
```

### Output:

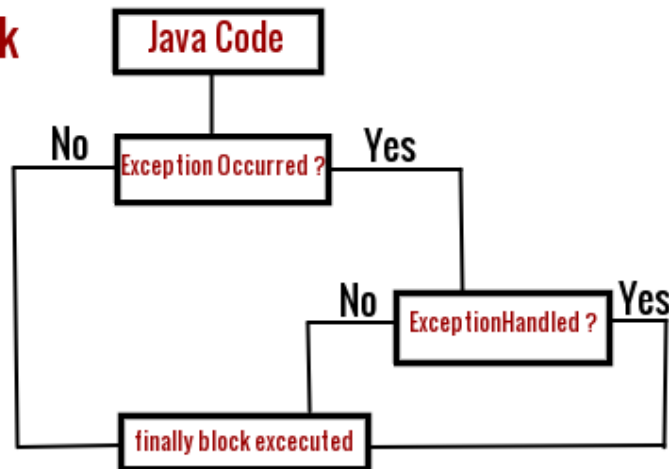
Arithmetic Exception occurs  
rest of the code

# Exception Handling (8)

## finally block

- **Java finally block** is a block that is used to execute important code such as closing connection, stream, etc
- Java finally block is **always executed** whether exception is handled or not

### Finally Block



```
Connection conn= null;
try {
    conn= get the db conn;
    //do some DML/DDDL
} catch(SQLException ex) {
} finally {
    conn.close();
}
```



3

# Checked And Unchecked Exception

# Checked Exceptions

## Introduction

- The **compiler checks them during compilation** to see whether the programmer has handled them or not
- If these exceptions are not handled/declared in the program, **it will give compilation error**
- Examples of Checked Exceptions
  - `NoSuchFieldException`
  - `SQLException`
  - etc

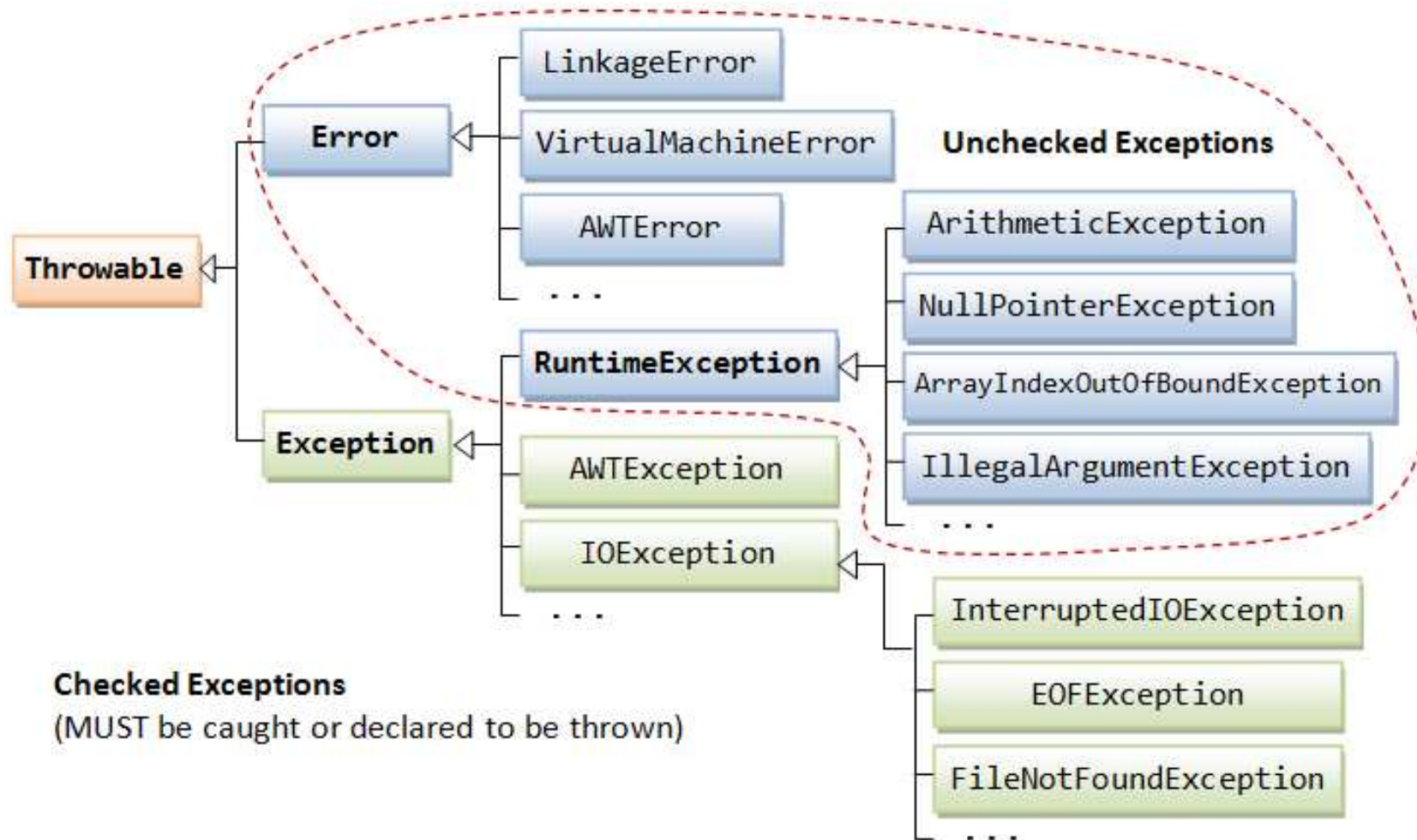
# Unchecked Exceptions

## Introduction

- The **compiler do not check whether** the programmer has handled them or not but it's the duty of the programmer to handle these exceptions and provide a safe exit
- If these exceptions are not handled/declared in the program, **it will not give compilation error**
- Examples of Checked Exceptions
  - `ArithmeticException`
  - `ArrayIndexOutOfBoundsException`
  - `NullPointerException`, etc

# Exception Classes

## Hierarchy of the Exception classes



4

# Throw and Throws

# throw Keyword (1)

## Introduction

- The Java throw keyword is used to explicitly **throw** an exception
- We can throw either **checked** or **unchecked** exception in java by throw keyword
- The throw keyword is mainly used to **throw custom** exception
- Syntax

```
throw exception;
```

# throw Keyword (2)

## Example

```
public static void isAge(int age) {  
    if (age < 18) {  
        throw new ArithmeticException("Not valid");  
    } else {  
        System.out.println("welcome to vote");  
    }  
}
```

```
public static void main(String[] args) {  
    try {  
        Validator.isAge(15);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

### Output:

```
java.lang.ArithmeticException: Not valid  
at rs.training.utils.Validator.isAge(Validator.java:20)  
at rs.training.main(Program.java:9)
```

# throws Keyword (1)

## Introduction

- The **Java throws keyword** is used to declare an exception
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained
- Syntax

```
return_type method_name() throws Exception_Class_Name {  
    // method code  
}
```



# throws Keyword (2)

## Example

- Creating a method that **throws a checked** exception

```
public void readBook(String file) throws FileNotFoundException {//#1
    boolean found = findFile(file);
    if (!found)
        throw new FileNotFoundException("Missing file"); //#2
    else {
        // code to read file
    }
}

public boolean findFile(String file) {
    // code to return true if file can be located
}
```

# throws Keyword (3)

## Example

- In which
  - **#1:** The throws statement **indicates** that this method can throw **FileNotFoundException**
  - **#2:** If file can't be found, code creates and **throws an object** of **FileNotFoundException** by using the **throw** statement

```
public void readBook(String file) throws FileNotFoundException {  
    // #1  
    boolean found = findFile(file);  
    if (!found)  
        throw new FileNotFoundException("Missing file"); // #2  
    else {  
        // code to read file  
    }  
}
```

# throws Keyword (4)

## Example

- Using a method that throws a checked exception

```
void useReadFile(String name) {  
    try {  
        readBook(name);  
    } catch (FileNotFoundException e) {  
        // code  
    }  
}
```

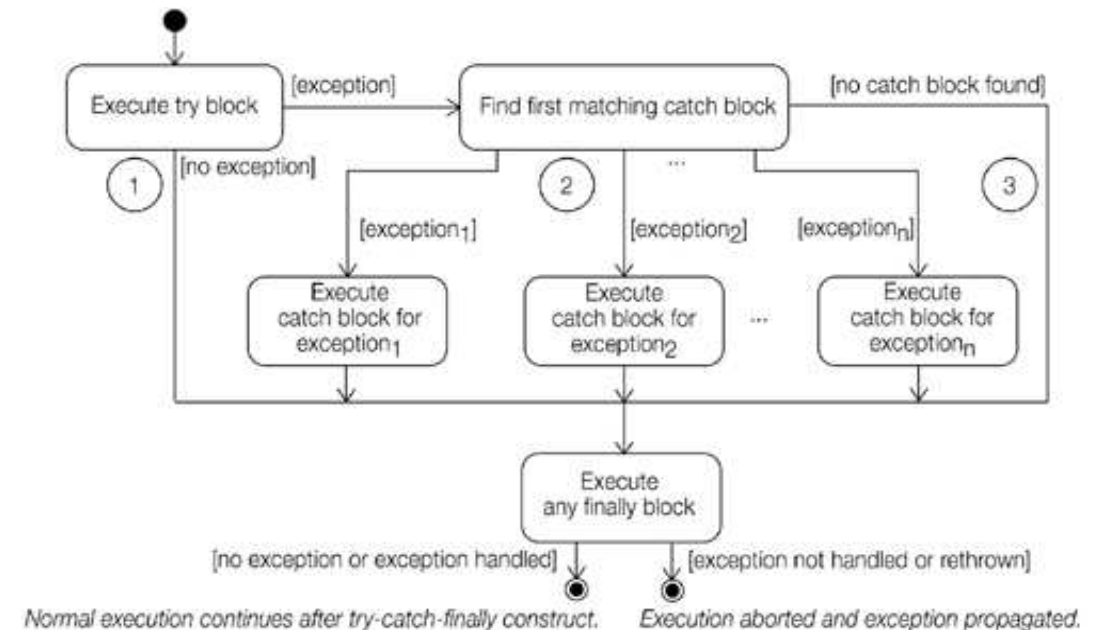
1

```
void useReadFile(String name) throws FileNotFoundException {  
    readBook(name);  
}
```

2

# Summary

- Java Exception
- Exception Handling
- Checked And Unchecked Exception
- Throw and throws keywords





# Thankyou!