# Lecture 9 - Oct 1

## Exceptions

***Execution Flows: Normal vs. Abnormal***
***Examples: Circle, Bank, ParseInt***

## Announcements/Reminders

- Today's class: notes template posted
- Priorities:
  - + Lab1 solution video to be released
  - + Lab2 to be released
- ProgTest1
  - + guide released
  - + PracticeTest1 released
  - + In-Person Review Session at 2 PM, Friday, Oct 3 (CLH C)

# Catch-or-Specify Requirement: Execution Flows (1)

**Scenario**: Current caller chooses to **catch/handle** the exception.

## Normal Flow of Execution

```
① .. /* before, ouside try-catch block */
  try {
② o.m(...); /* may throw SomeException */
③ ... /* rest of try-block */
  }
  catch (SomeException se) {
    ... /* rest of catch-block */
  }
④ .. /* after, ouside try-catch block */
```

exception does not occur

end of try

When the exception does not occur

bypassed ∵ exception did not occur.

## Abnormal Flow of Execution

```
① .. /* before, ouside try-catch block */
  try {
② o.m(...); /* may throw SomeException */
X ... /* rest of try-block */
  }
③ catch (SomeException se) {
④ ... /* rest of catch-block */
  }
⑤ .. /* after, ouside try-catch block */
```

exception occurred.

When the exception occurs

bypassed ∵ exception occurred.

# Catch-or-Specify Requirement: Execution Flows (2)

**Scenario**: Caller chooses to **specify/propogage** the exception.

**Normal** Flow of Execution

```
class C1 {
  void m1 throws SomeException {
    ... /* some code */
    ... /* some code */
    C2 o = new C2();
    o.m();
    ... /* some code */
    ... /* some code */
  }
}
```

exception did not occur

When the exception does not occur

exception did not disrupt the flow of exec

**Abnormal** Flow of Execution

caller

```
class C1 {
  void m1 throws SomeException {
    ... /* some code */
    ... /* some code */
    C2 o = new C2();
    o.m();
    ... /* some code */
    ... /* some code */
  }
}
```

callee

exception occurred

thrown C2.m

thrown C1.m1

C1.m1 terminates abnormally here

exception thrown disrupted the flow.

When the exception occurs

# Error Handling via Exceptions: Circles (Version 2)

```
public class InvalidRadiusException extends Exception {
  public InvalidRadiusException(String s) {
    super(s);
  }
}
```

**Test Case:**

User enters -5

Then user enters 10

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

```
public class CircleCalculator2 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean inputRadiusIsValid = false;
    while (!inputRadiusIsValid) {
      System.out.println("Enter a radius:");
      double r = input.nextDouble();
      Circle c = new Circle();
      try {
        c.setRadius(r);
        inputRadiusIsValid = true;
        System.out.print("Circle with radius " + r);
        System.out.println(" has area: "+ c.getArea()); }
      catch (InvalidRadiusException e) { System.out.print("Try again!"); }
    } } }
```

Console

```
Enter a radius:
-5
Try again
Enter a radius:
Circle with area ...
```

Code.ser

CCZ.main

# Error Handling via Exceptions: Circles (Version 1)

```
public class InvalidRadiusException extends Exception {
  public InvalidRadiusException(String s) {
    super(s);
  }
}
```

**Test Case 1:**

User enters 10

**Test Case 2:**

User enters -5

```
class Circle {
  double radius;
  Circle() { /* radius defaults to 0 */ }
  void setRadius(double r) throws InvalidRadiusException {
    if (r < 0) {
      throw new InvalidRadiusException("Negative radius.");
    }
    else { radius = r; }
  }
  double getArea() { return radius * radius * 3.14; }
}
```

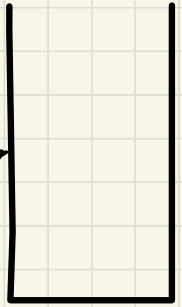
| caller | callee |
|---|---|
| | |

```
class CircleCalculator1 {
  public static void main(String[] args) {
    Circle c = new Circle();
    try {
      c.setRadius(-10);
      double area = c.getArea();
      System.out.println("Area: " + area);
    }
    catch(InvalidRadiusException e) {
      System.out.println(e);
    }
  } }
```

Caller?

Callee?

call stack



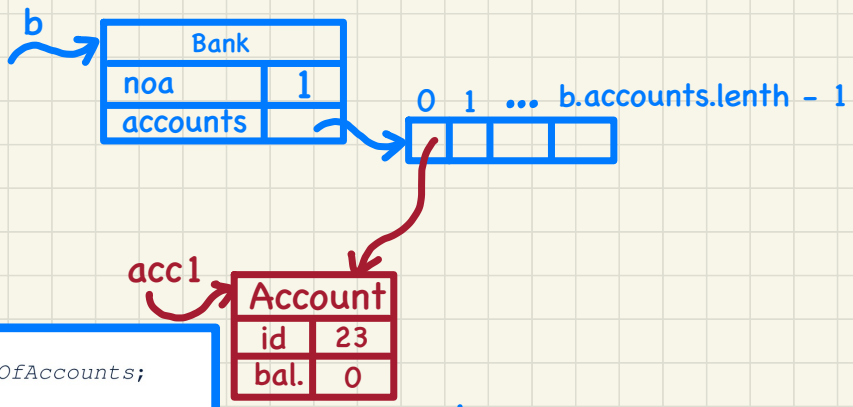
# Error Handling via Exceptions: Banks

```
public class InvalidTransactionException extends Exception {
  public InvalidTransactionException(String s) {
    super(s);
  }
}
```

```
class Account {
 int id; double balance;
 Account() { /* balance defaults to 0 */ }
 void withdraw(double a) throws InvalidTransactionException {
   if (a < 0 || balance - a < 0) {
     throw new InvalidTransactionException("Invalid withdraw."); }
   else { balance -= a; }
 }
}
```

```
class Bank {
 Account[] accounts; int numberOfAccounts;
 Account(int id) { ... }
 void withdraw(int id, double a)
     throws InvalidTransactionException {
   for(int i = 0; i < numberOfAccounts; i ++) {
     if(accounts[i].id == id) {
       accounts[i].withdraw(a);
     }
   } /* end for */ }
```

```
class BankApplication {
 pubic static void main(String[] args) {
  Bank b = new Bank();
  Account acc1 = new Account(23);
  b.addAccount(acc1);
  Scanner input = new Scanner(System.in);
  double a = input.nextDouble();
  try {
    b.withdraw(23, a);
    System.out.println(acc1.balance); }
  catch (InvalidTransactionException e) {
    System.out.println(e); } } }
```




Account.withdraw

Bank.withdraw

BA.main

accounts[i].withdraw

Account[]

Account(

Exercise.

**Test Case:**

User enters **-5000000**

# More Example: Multiple Catch Blocks

Assumption
All the customized exceptions are unrelated to each other.

```
double r = ...;
double a = ...;
try{
①Bank b = new Bank();
③b.addAccount(new Account(34));
②b.deposit(34, 100);
④b.withdraw(34, a);
  Circle c = new Circle();
  c.setRadius(r);
  System.out.println(r.getArea());
}
catch(NegativeRadiusException e) {
  System.out.println(r + " is not a valid radius value.");
  e.printStackTrace();
}
catch(InvalidTransactionException e) {
⑤System.out.println(r + " is not a valid transaction value.");
⑥e.printStackTrace();
}
```

-5M

throw ITE

-5M → may throw ITE

may throw NRE

**Test Case 1:**
a: -5000000
r: 23

**Test Case 2:**
a: 100
r: -5

→ order of catch blocks does not matter.

# More Example: Parsing Strings as Integers

false
vI

```
Scanner input = new Scanner(System.in);
boolean validInteger = false;
while (!validInteger) {
    System.out.println("Enter an integer:");
    String userInput = input.nextLine();
    try {
        int userInteger = Integer.parseInt(userInput);
        validInteger = true;
    }
    catch(NumberFormatException e) {
        System.out.println(userInput + " is not a valid integer.");
        /* validInteger remains false */
    }
}
```

**Test Case:**

User Enters: twenty-three

User Then Enters: 23

"twenty-three" → NFE thrown

"23" → NFE not thrown

Enter an integer:
twenty-three

Try again

Enter an integer:
23

# Review: Specify-or-Catch Principle

**Approach 1 – Specify**: Indicate in the method signature that a specific exception might be thrown.

**Example 1:** Method that throws the exception

```
class C1 {
  void m1(int x) throws ValueTooSmallException {
    if(x < 0) {
      throw new ValueTooSmallException("val " + x);
    }
  }
}
```

excep. originated

**Example 2:** Method that calls another which throws the exception

```
class C2 {
  C1 c1;
  void m2(int x) throws ValueTooSmallException {
    c1.m1(x);
  }
}
```

caller may throw VTSE

# Review: Specify-or-Catch Principle

**Approach 2 – Catch**: Handle the thrown exception(s) in a try-catch block.

```
class C3 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int x = input.nextInt();
    C2 c2 = new c2();
    try {
      c2.m2(x);
    }
    catch(ValueTooSmallException e) { ... }
  }
}
```

no need to specify VTSE.

callee may throw VTSE

# A Class for Bounded Counters

```
public class Counter {
  public final static int MAX_VALUE = 3;
  public final static int MIN_VALUE = 0;
  private int value;
  public Counter() {
    this.value = Counter.MIN_VALUE;
  }
  public int getValue() {
    return value;
  }
  ... /* more later!
```

valid values for counter: 0, 1, 2, 3

```
/* class Counter */
  public void increment() throws ValueTooLargeException {
    if(value == Counter.MAX_VALUE) {
      throw new ValueTooLargeException("counter value is " + value);
    }
    else { value ++; }
  }

  public void decrement() throws ValueTooSmallException {
    if(value == Counter.MIN_VALUE) {
      throw new ValueTooSmallException("counter value is " + value);
    }
    else { value --; }
  }
}
```

* NTSE and NTLE not expected → if any exception occurred → fail the test

# Coming Up with Test Cases: A Single, Bounded Variable

** NTSE or NTLE expected → if the expected exception occurred → pass the test

Boundries: 0 3

Counter.MIN_VALUE <= c.value <= Counter.MAX_VALUE