Lecture 9 - June 3

Exceptions, TDD with JUnit

Example: Exceptions, Loops, Boolean Var.
Example: Converting Strings to Integers
Bounded Counter: Deriving Test Cases

Announcements/Reminders

- Today's class: notes template posted
- Changes on test dates (still during **enrolled** session):
  + ProgTest1: JUN 6 -> **JUN 13** (time+ to study Lab1 sol)
  + ProgTest2: JUL 4 -> **JUL 11** (time+ to study Lab3 sol)
- To be released this week:
  + ProgTest1 guide (policies & requirements)
  + PracticeTest1 & Survey on Review Session Time
  + ProgTest0 marks & feedback (or MON, JUN 9 latest)
- Priorities:
  + Lab1 solution, Lab2
  + Slides on **Classes and Objects**
  + Slides on **Exceptions**

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) { print("Try again!"); }
    } } }
```

Console

```
Enter a radius:
-5
Try again!
Enter a radius:
10 Circle with r.10 has
```

Circle.sR
CC2.main
cvpa →314.0.

Error Handling via Exceptions: Banks

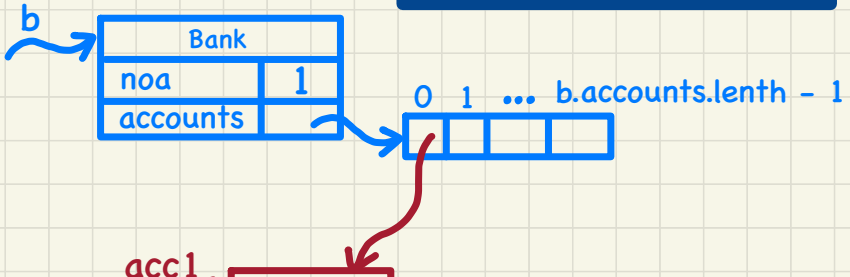
Test Case:
User enters **-5000000**

```
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; }
  }
}
```

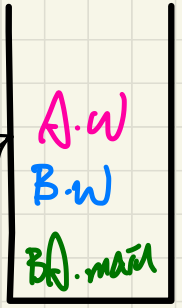
```
class Bank {
  Account[] accounts; int numberOfAccounts;
  Account(int id) { ... }
  void withdraw(int id, double a)
      throws InvalidTransactionException {
    for(int i = 0; i < numberOfAccounts; i ++) {
      if(accounts[i].id == id)
        accounts[i].withdraw(a);
    } /* end for */ }
```

```
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); } } }
```



caller	callee

call stack



More Example: Multiple Catch Blocks

Assume: customized exceptions (e.g. ITE, NRE) are unrelated to each other ⇒ their catch blocks can be ordered in any way.

```
1  double r = 23;
2  double a = -5M;
3  try{
4    Bank b = new Bank();
5    b.addAccount(new Account(34));
6    b.deposit(34, 100);
7    b.withdraw(34, a);        // may throw ITE
     Circle c = new Circle();
     c.setRadius(r);           // may throw NRE
     System.out.println(r.getArea());
   }
   catch(NegativeRadiusException e) {
     System.out.println(r + " is not a valid radius value.");
     e.printStackTrace();
   }
8  catch(InvalidTransactionException e) {
9    System.out.println(r + " is not a valid transaction value.");
10   e.printStackTrace();
   }
11 // outside the try-catch block
```

Test Case 1:

a: -5000000

r: 23

Test Case 2:

a: 100

r: -5

More Example: Parsing Strings as Integers

```
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" → throw NFE
"23" → NFE not thrown

/* exit from loop */

Enter an integer:
twenty-three
twenty-three invalid
Enter an integer:
23

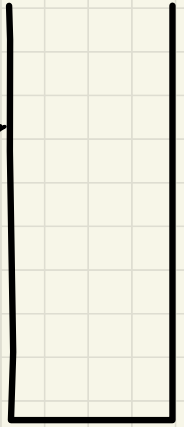
Error Handling via Console Messages: Circles

```
1  class Circle {
2    double radius;
3    Circle() { /* radius defaults to 0 */ }
4    void setRadius(double r) {
5      if ( r < 0 ) { System.out.println( "Invalid radius." ); }
6      else { radius = r; }
7    }
8    double getArea() { return radius * radius * 3.14; }
9  }
```

Caller?

Callee?

call stack



```
1  class CircleCalculator {
2    public static void main(String[] args) {
3      Circle c = new Circle();
4      c.setRadius( -10 );
5      double area = c.getArea();
6      System.out.println("Area: " + area);
7    }
8  }
```

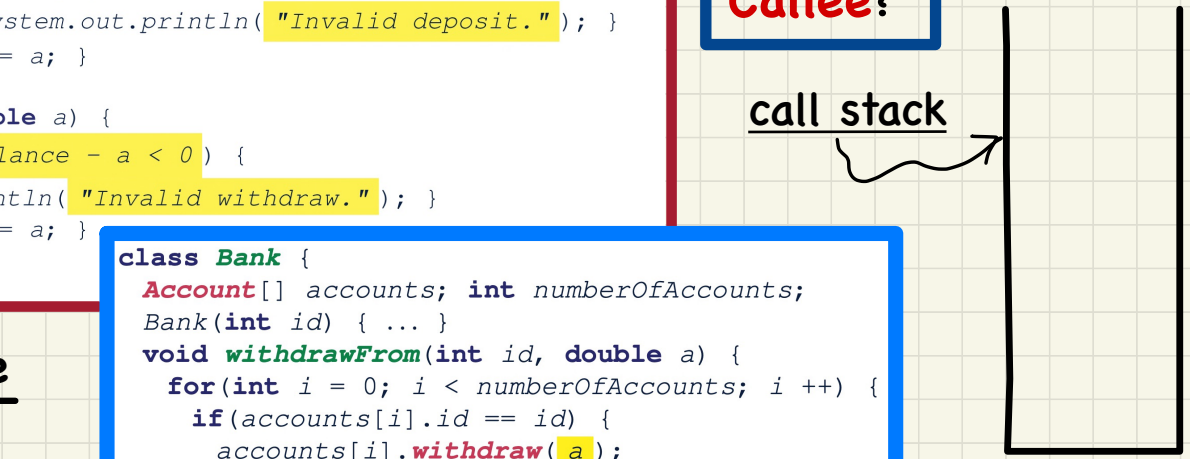
Error Handling via Console Messages: Banks

```
class Account {
  int id; double balance;
  Account(int id) { this.id = id; /* balance defaults to 0 */ }
  void deposit(double a) {
    if ( a < 0 ) { System.out.println( "Invalid deposit." ); }
    else { balance += a; }
  }
  void withdraw(double a) {
    if ( a < 0 || balance - a < 0 ) {
      System.out.println( "Invalid withdraw." ); }
    else { balance -= a; }
  }
}
```

Caller?

Callee?

call stack



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

```
class BankApplication {
  pubic static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    Bank b = new Bank(); Account acc1 = new Account(23);
    b.addAccount(acc1);
    double a = input.nextDouble();
    b.withdrawFrom(23, a );
    System.out.println("Transaction Completed.");
}
```

context	caller	callee

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

origin of exception

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

Example 2: Method that calls another which throws the exception

calls may choose to specify

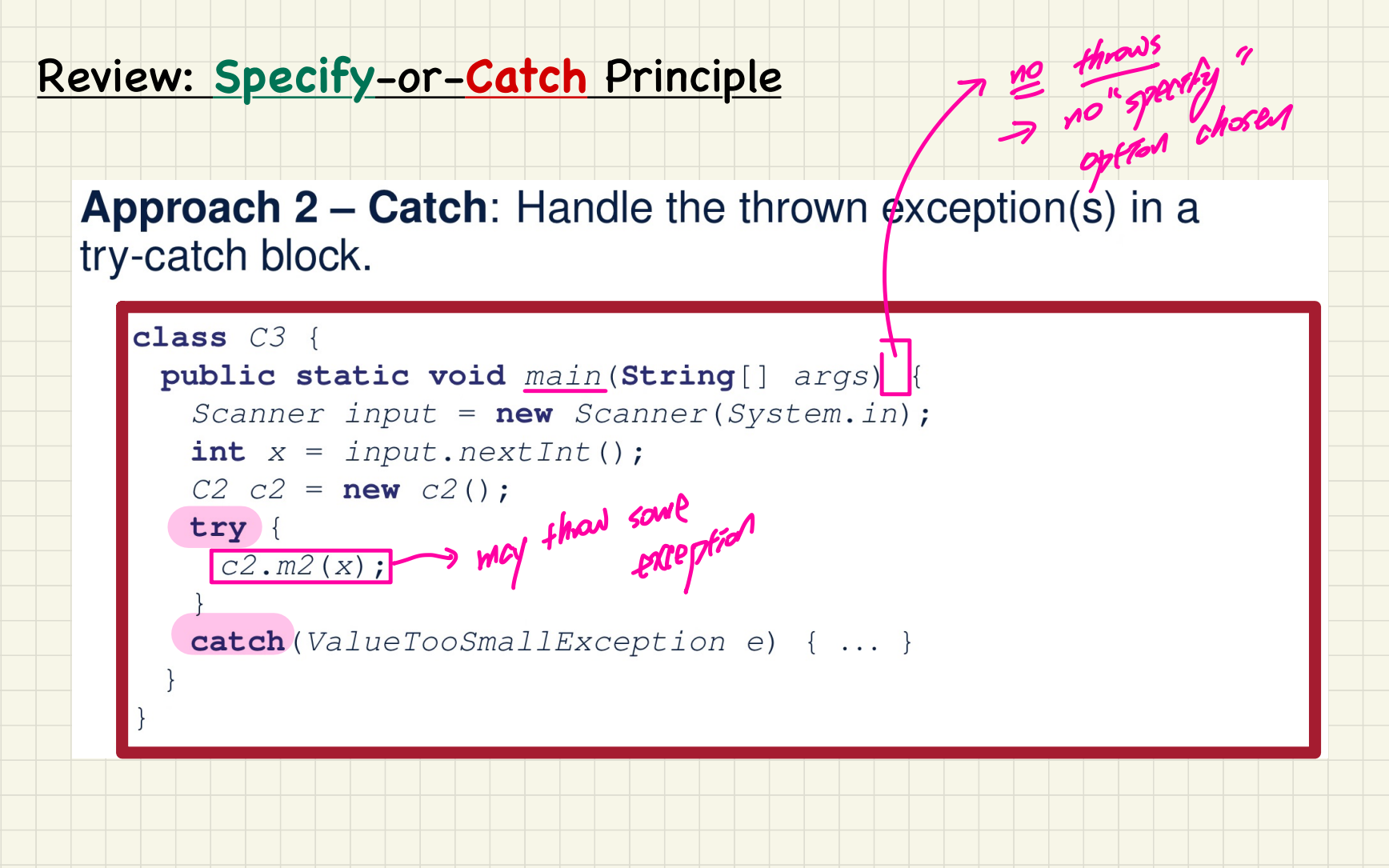
```
class C2 {
  C1 c1;
  void m2(int x) throws ValueTooSmallException {
    c1.m1(x);
  }
}
```

may throw some exception

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) { ... }
  }
}
```



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!
```

0

specify

```
/* 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 --; }
  }
}
```

specify

① normal scenarios to test ② abnormal scenarios to test

Coming Up with Test Cases: A Single, Bounded Variable

① exception occurred → fail ; exception not occurred → pass

Boundries: ② exception occurred → pass not occurred → fail
(expected)

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