Lecture 9 - Oct 2

Exceptions

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

Announcements/Reminders

- Today's class: notes template posted
- Priorities:
  - + Lab1 solution video released
  - + Lab2 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 not thrown

When the exception does not occur

bypassed :(no exception occurred

Abnormal Flow of Execution

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

exception thrown

When the exception occurs

rest of try block 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 {
    1 ... /* some code */
    2 ... /* some code */
    3 C2 o = new C2();
    4 o.m();
    5 ... /* some code */
    6 ... /* some code */
  }
}
```

exception not occurred

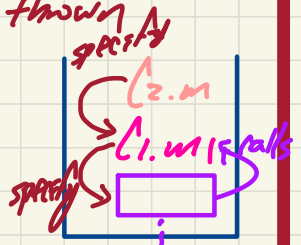
exception disrupted the flow and thrown to the caller of C1.m1

When the exception does not occur

Abnormal Flow of Execution

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

exception thrown



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; }
}
```

terminates setR meth. abnormally

terminates setR normally

```
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!"); }
    } } }
```

IRE thrown

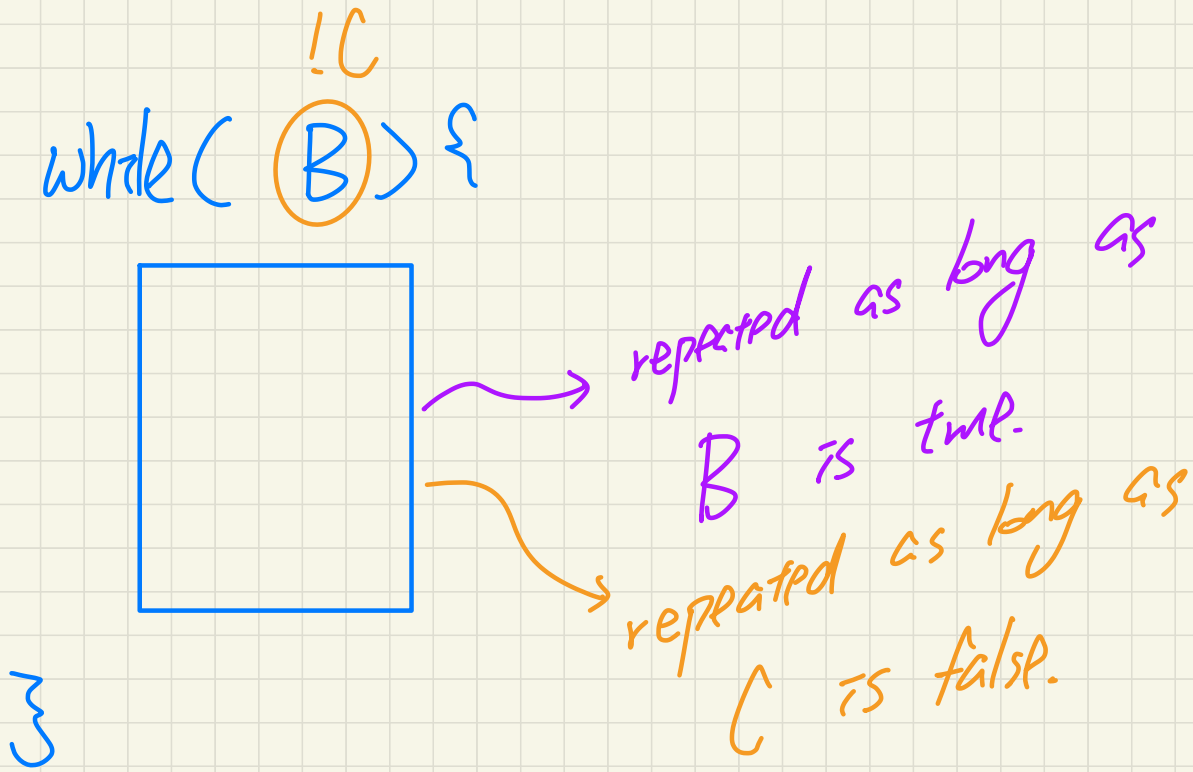
IRE not thrown

Console

```
Enter a radius:
-5
Try Again
Enter a radius:
10 Area is ...
```

C.setR

CC2.main


!C
while (B) {
repeated as long as B is true.
repeated as long as C is false.
}

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; }
}
```

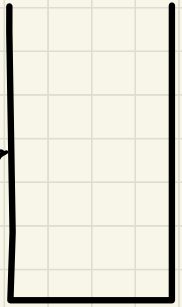
caller	callee

```
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



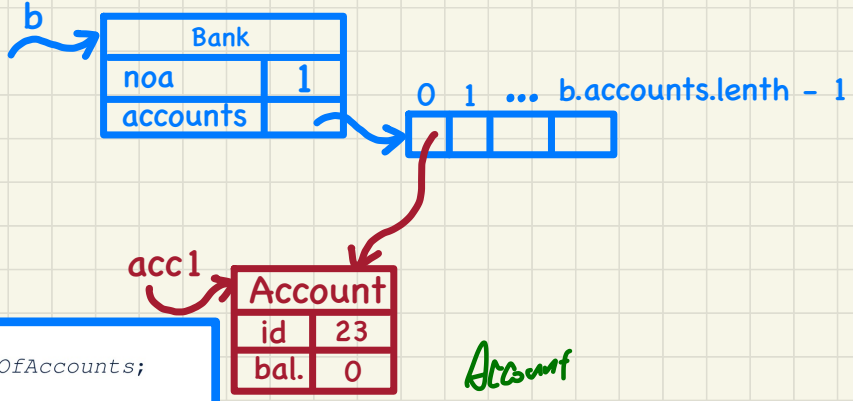
Error Handling via Exceptions: Banks

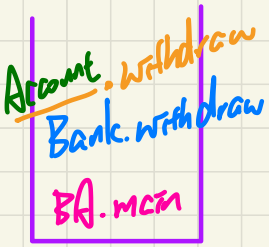
```
public class InvalidTransactionException extends Exception {
  public InvalidTransactionException(String s) {
    super(s);
  }
}
```

```
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); } } }
```





Test Case:

User enters **-5000000**

More Example: Multiple Catch Blocks

Assumption: Customized exceptions are unrelated to each other
↳ their catch blocks can be arranged in any order

```
double r = ...;
double a = ...;
try{
Bank b = new Bank();
b.addAccount(new Account(34));
b.deposit(34, 100);
b.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) {
System.out.println(r + " is not a valid transaction value.");
e.printStackTrace();
}
```

1 2 3 4 5 6 7 8 9 10

-5 100

may throw ITE → not thrown

-5 → may throw NRE → thrown

Test Case 1:

a: -5000000

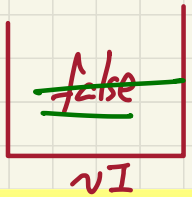
r: 23

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

NFE thrown
"twenty-three"

Enter an integer:
twenty-three
Not valid
Enter an integer:
23

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!
```

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

Coming Up with Test Cases: A Single, Bounded Variable

Boundries:

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