Lecture 8 - Oct. 1

Exceptions

CoS Req.: Exec. Flow vs. Call Stack
To Handle or Not to Handle (V2 - V4)
More Examples on Exceptions
Exceptions vs. Console Tester

Announcements/Reminders

- Written Test 1 tomorrow (Wednesday)
- WrittenTest1 review session recording released
- Lab1 due this Friday
- Mockup Programming Test grading tests released
- Mockup Programming Test feedback to be released

Catch-or-Specify Requirement: Execution Flows

Normal Flow of Execution

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

excep not occurred

X bypassed

When the exception does not occur

Abnormal Flow of Execution

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

excep. occurred

X bypassed

When the exception occurs

Catch-or-Specify Requirement: Call Stack

Q1. Origin of Exception: $C1.m1$

Q2. Methods subject to CoS Req:

$C1.m1 \sim C_{i-1}.m_{i-1}$

Q3. Methods free from CoS Req:

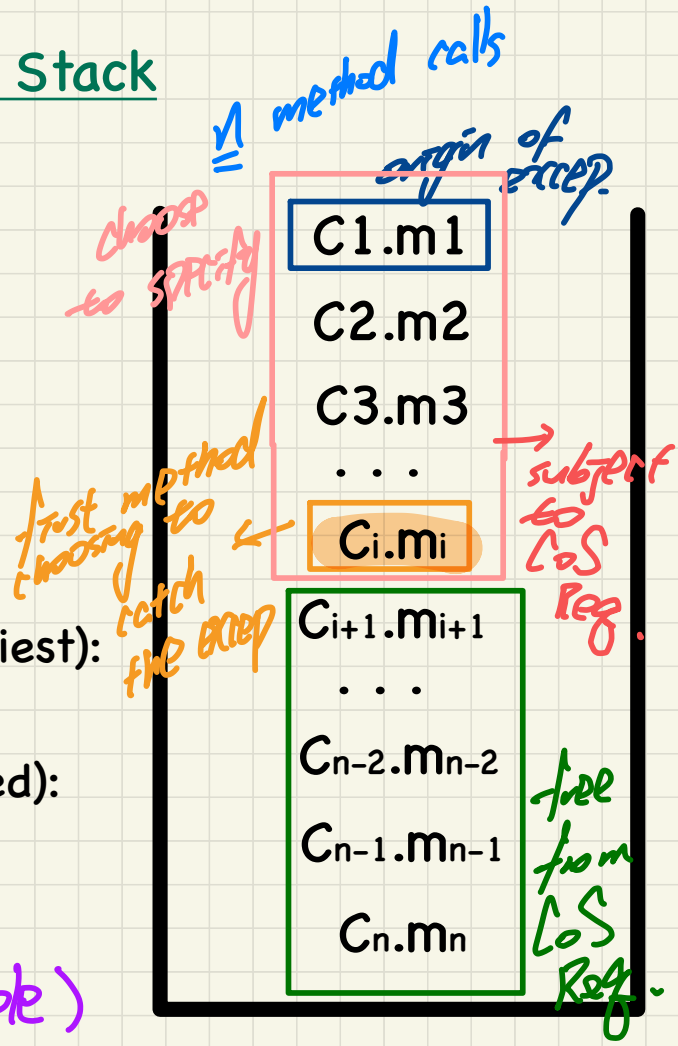
$C_{i+1}.m_{i+1} \sim Cn.mn$

Q4. Extreme Case 1 (exception handled earliest):

$C2.m2$

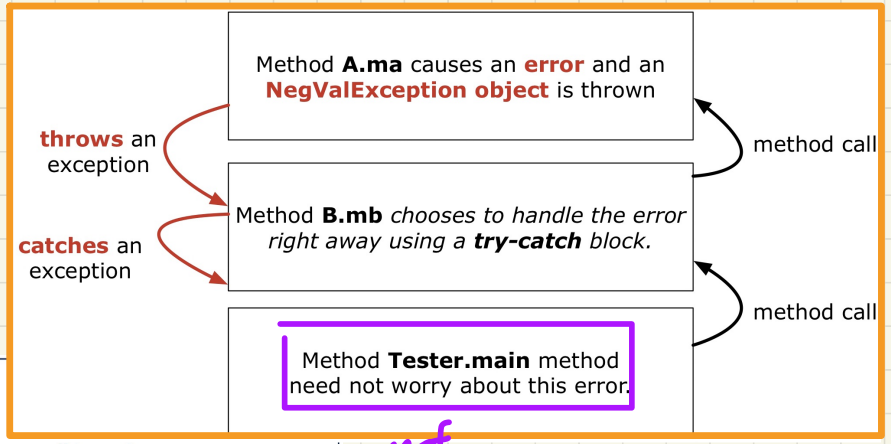
Q5. Extreme Case 2 (exception never handled):

All methods choose to specify (excep. thrown to console)



Version 1:

Handle the Exception in B.mb




```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

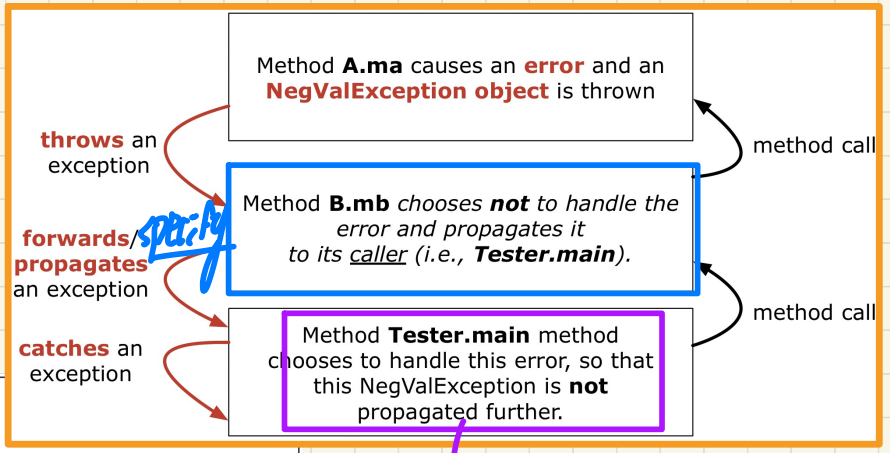
```
class B {
  mb(int i) {
    A oa = new A();
    try { oa.ma(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i); /* Error, if any, would have been handled in B.mb.
  } }
```

not subject to CoS Req.

Version 2:

Handle the Exception in Tester.main



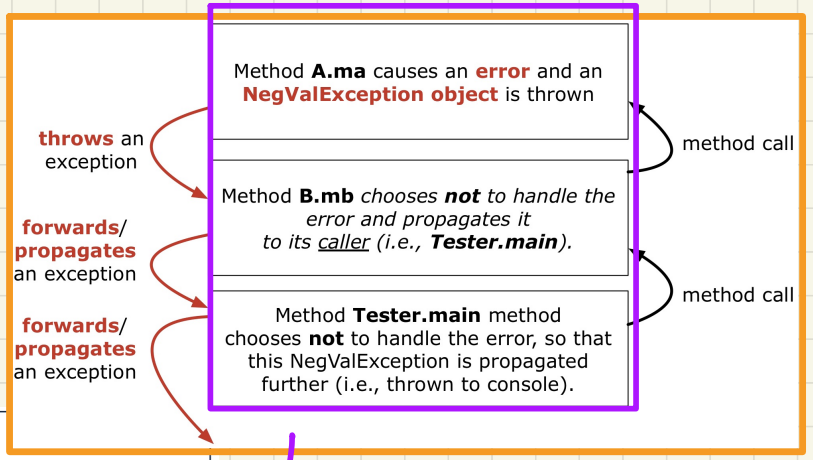

```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    try { ob.mb(i); }
    catch(NegValException nve) { /* Do something. */ }
  } }
```

Version 3:

Handle in Neither Classes on Call Stack



all subject to LoS req.

```
class A {
  ma(int i) throws NegValException {
    if(i < 0) { throw new NegValException("Error."); }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) throws NegValException {
    A oa = new A();
    oa.ma(i);
  } }
```

```
class Tester {
  public static void main(String[] args) throws NegValException {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    ob.mb(i);
  } }
```

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; }
}
```

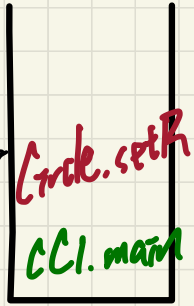
caller	callee

```
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: Circles (Version 2)

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

Test Case:

User enters -5

Then user enters 10

Exercise!

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

keep prompting for a non-neg. radius

may throw IRE

except. not occurred ⇒ ready to exit loop

except. occurred!

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; }
  }
}
```

specify

-5M

```
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 */ }
```

specify

-5M

Account

```
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); } } }
```

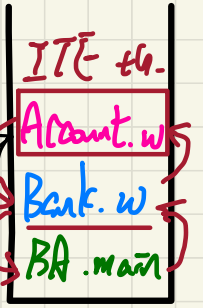
-5M -5M

catch option.

b → Bank { noa: 1, accounts } → 0 1 ... b.accounts.lenth - 1

acc1 → Account { id: 23, bal.: 0 }

call stack



More Example: Multiple Catch Blocks

```
1  double r = ...;
2  double a = ...;
3  try{
4    Bank b = new Bank();
5    b.addAccount(new Account(34));
6    b.deposit(34, 100);
7    b.withdraw(34, a);
8    Circle c = new Circle();
9    c.setRadius(r);
10   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();
   }
```

may throw ITE

may throw NRE

Test Case 1:

a: -5000000

r: 23

Test Case 2:

a: 100

r: -5

execute!

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" → trigger NFE

"23" → does not trigger NFE

may throw NFE

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  }
```

a bad alternative to throwing an excep.

no CoS req. enforced

any more

Caller?

Callee?

call stack

C.sR

CC.main

```
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  }
```

no excep. thrown (only a msg printed to console)

still executed despite Error.

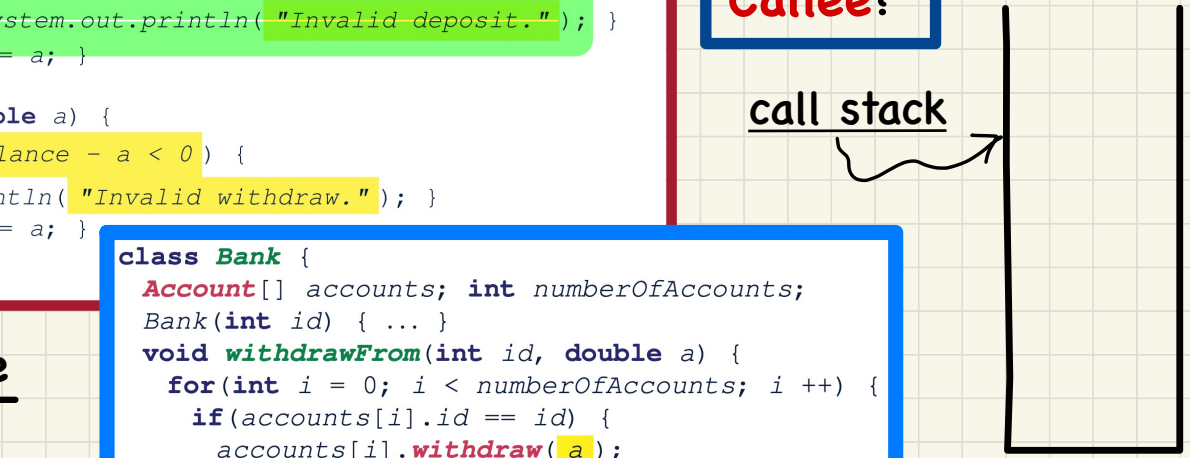
Error Handling via Console Messages: Banks

```
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