Lecture 2a

Part A

Exceptions –
Caller vs. Callee in a Method Invocation

Caller vs. **Callee**

Caller → party calling another method

Callee → party being called by another method

- **caller** is the **client** using the service provided by another method.
- **callee** is the **supplier** providing the service to another method.

```
class C1 {
  void m1() {
    C2 o = new C2();
    o.m2(); /* static type of o is C2 */
  }
}
```

C1 → context of method call (caller)

context of a method call/invocation
1. class
2. method

o.m2(): o → context object

callee: class: (type of o) C2
method: m2

Q: Can a method be a caller and a callee simultaneously?

```
class C3 {
  void m3() {
    C1 o = new C1();
    o.m1();
  }
}
```

callee: C1.m1

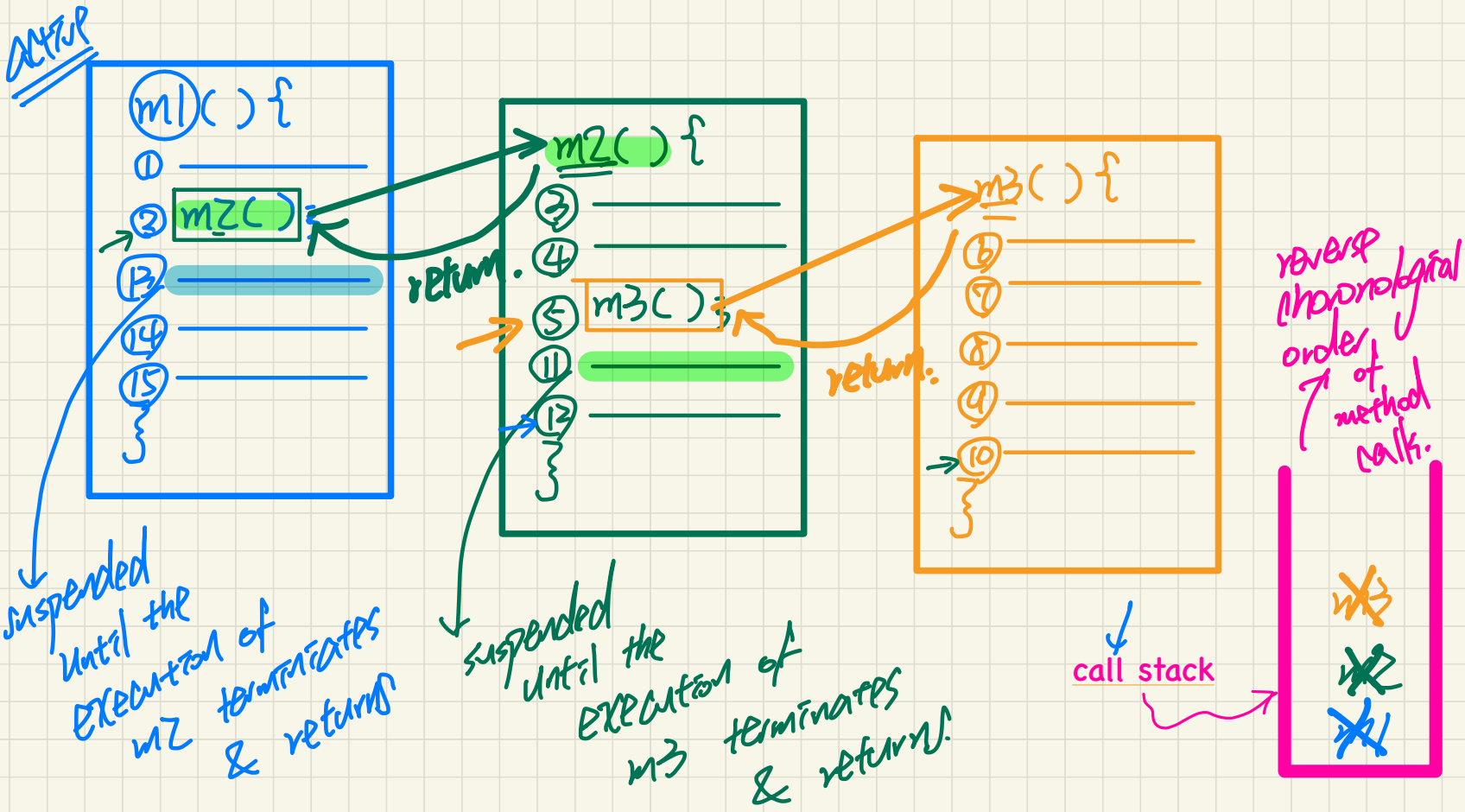
Can C2.m2 be a caller as well?
YES: make some method call m → C2.m2

Lecture 2a

Part B

Exceptions –
Visualizing a Method Call Chain as a Stack

Visualizing a Call Chain using a Stack



Lecture 2a

Part C

Exceptions – Error Handling via Console Messages

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?

```
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  }
```

call stack

CircleCalculator.main

Console
Invalid radius.
Area 0.0

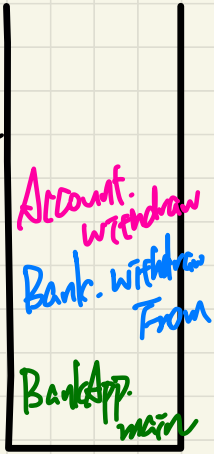
Error Handling via Console Messages: Banks

```
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;
  Account(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.");
  }
```

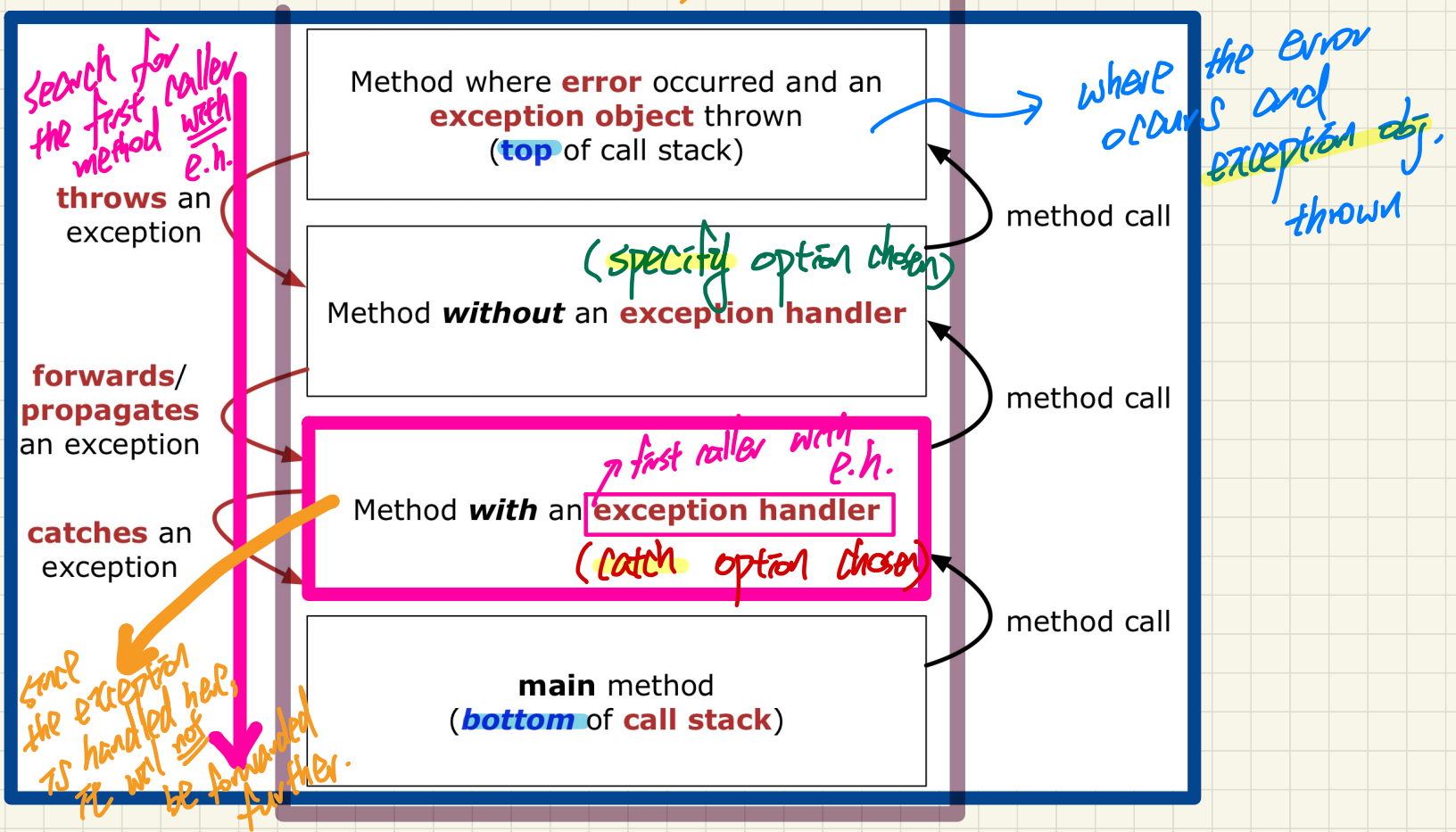
context	caller	callee
BankApp.	main	Bank. withdraw From
Bank	withdraw From	Account. withdraw
Account	withdraw	✗

Lecture 2a

Part D

Exceptions -
When an Exception is Thrown,
The Catch-or-Specify Requirement

What to Do When an Exception is Thrown: Call Stack



Catch-or-Specify Requirement

The "Catch" Solution: A `try` statement that *catches* and *handles* the *exception* (**without** propagating that exception to the method's *caller*).

```
main(...) {
  Circle c = new Circle();
  try {
    c.setRadius(-10);
  }
  catch(NegativeRaidusException e) {
    ...
  }
}
```

callee throws an exception upon an invalid input value

The "Specify" Solution: A method that specifies as part of its *header* that it may (or may not) *throw* the *exception* (which will be thrown to the method's *caller* for handling).

```
class Bank {
  Account[] accounts; /* attribute */
  void withdraw (double amount)
      throws InvalidTransactionException {
    ...
    accounts[i].withdraw(amount);
    ...
  }
}
```

header of method

callee may throw an exception upon an invalid amount

Recap of Exceptions

- Catch-or-Specify Requirement

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 */
```

no exception was thrown

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 */
```

exception was thrown

When the exception occurs

Lecture 2a

Part E

Exceptions –
Example: To Handle or Not to Handle?

Example: To Handle or Not To Handle?

```
class A {
  ma(int i) {
    if(i < 0) { /* Error */ }
    else { /* Do something. */ }
  } }
```

```
class B {
  mb(int i) {
    A oa = new A();
    oa.ma(i); /* Error occurs if i < 0 */
  } }
```

```
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); /* Where can the error be handled? */
  } }
```

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

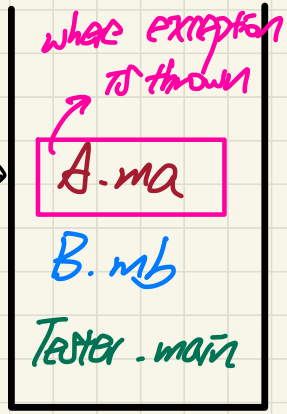
context	caller	callee
Tester	main	B.mb
B	mb	A.ma
A	ma	✗

Version 1:
Handle it in `B.mb`

Version 2:
Pass it from `B.mb` and handle it in `Tester.main`

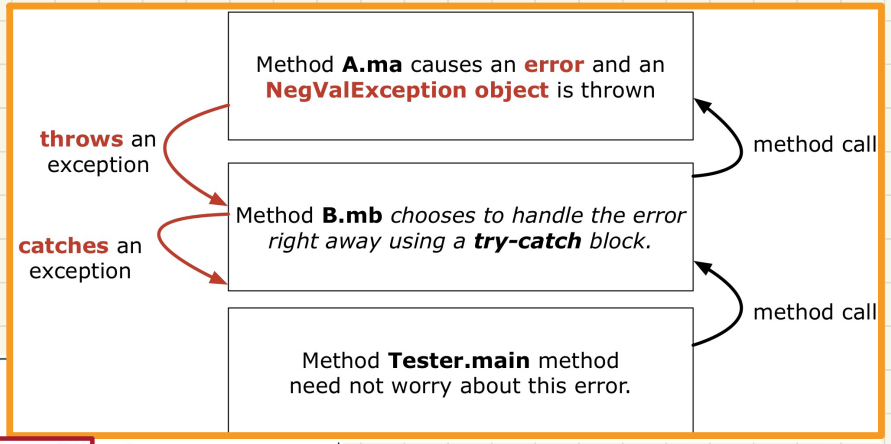
Version 3:
Pass it from `B.mb`, then from `Tester.main`, then throw it to the console.

call stack



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. */ }
  } }
```

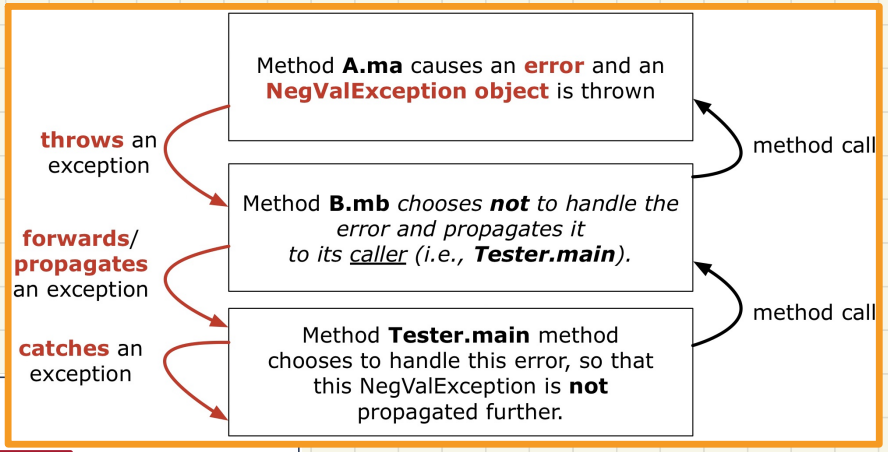
callee throws NVE

```
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.
  } }
```

B.mb already handles the NVE, so Tester.main does not need to catch or specify it.

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. */ }
  } }
```

↳ exception thrown

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

callee throws an exception, but in B.mb, we choose to specify it.

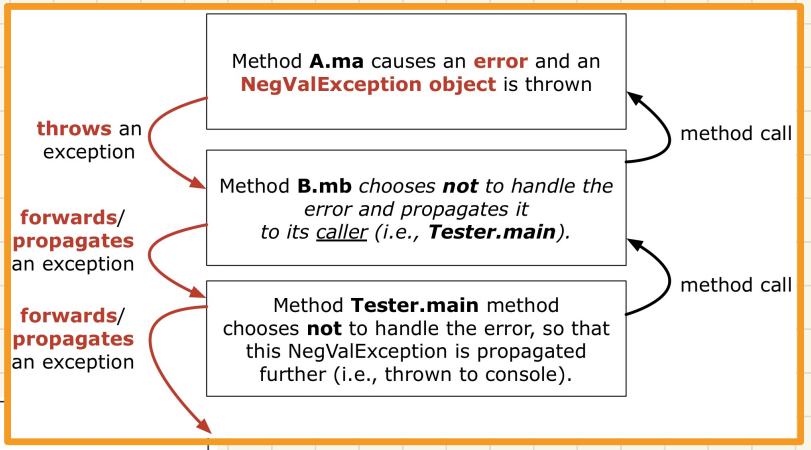
```
class Tester {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int i = input.nextInt();
    B ob = new B();
    try { ob.mb(i); }
    catch(NegValException nve) { /* Do something. */ }
  }
```

callee specifies the NVE may be thrown.

Consequence:
caller of B.mb will be forced to either catch or specify the NVE.
Consequence: NVE will not be thrown to caller.

Version 3:

Handle in Neither Classes on Call Stack




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

↳ where the exception is originated

```
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);
  } }
```

Lecture 2a

Part F

Exceptions –
Error Handling via Exceptions

Error Handling via Exceptions: Circles (Version 1)

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

```java
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; }
}
```

specify

where the IRE is originated

```java
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);
    }
  }
}
```

once the exception is handled here, it will not be propagated further.

Error Handling via Exceptions: Circles (Version 2)

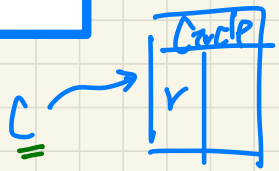
```
public class InvalidRadiusException extends Exception {
  public InvalidRadiusException(String s) {
    super(s);
  }
}
```

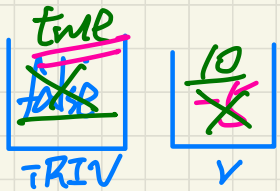
Test Case:

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; }
}
```





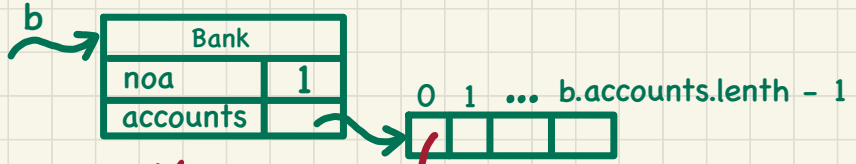

```
public class CircleCalculator2 {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean inputRadiusIsValid = false;
    while (!inputRadiusIsValid) {
      System.out.println("Enter a radius:");
      double r = input.nextDouble();
      Circle c = new 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!"); }
    } } }
```

-5 10 IRE 10 3.14

Enter a radius:
-5
Try again!
Enter a radius: 10
Circle with radius 10 has area 3.14

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; }
  }
}
```

exception originated.

acc1

Account	
id	23
bal.	0

```
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 */ }
```

exception thrown from Account.withdraw

↳ since Bank.withdraw specifies it, it will be propagated to BankApp.main.

```
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

Lecture 2a

Part G

Exceptions – More Examples

More Example: Multiple Catch Blocks

```
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();
}
```

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

skip

true

VI

"twenty-three"

"23"

"23"

NFE

may throw NFE

reaching this line means the NFE did not occur → input string was successfully converted into int.

"twenty-three"