Department of Computer Science
Undergraduate Events

- **Events this week**
  - **Drop-In Resume Editing**
    - Date: Mon. Jan 11
    - Time: 11 am – 2 pm
    - Location: Rm 255, ICICS/CS
  - **Industry Panel**
    - Speakers: Managers from IBM, Microsoft, SAP, TELUS, Radical ...
    - Date: Tues. Jan 12
    - Time: Panel: 5:15 – 6:15 pm
      Networking: 6:15 – 7:15 pm
    - Location: DMP 110 for panel,
      X-wing ugrad lounge for networking

- **Tech Career Fair**
  - Date: Wed. Jan 13
  - Time: 10 am – 4 pm
  - Location: SUB Ballroom

- **Google Tech Talk**
  - Date: Wed, Jan 13
  - Time: 4 – 5 pm
  - Location: DMP 110

- **IBM Info Session**
  - Date: Wed, Jan 13
  - Time: 5:30 – 7 pm
  - Location: Wesbrook 100

- **Adminstrivia**
  - Lecture slides (day by day) are on the web:
  - Reminder: Assignment #1 is on the web site
    - Due Thursday, January 28th, 10:00pm
  - How was Lab1?
  - What do you think of the new room?
Where are we?

We’ve designed a single class

- Attributes
- Methods
- Class invariants (for attributes)
- Preconditions and postconditions (for methods)

Class Design
Handling Errors

You will be expected to:

- incorporate exception handling into the design of a method's contract
- trace code that makes use of exception handling
- write code to throw, catch or propagate an exception
- write code that uses a finally block
- write code to define a new exception class
- compare and contrast checked and unchecked exceptions
  - understand the consequence of using checked vs. unchecked exceptions

Reading:
2nd Ed: Chapter 15
3rd, 4th Ed: Chapter 11

Exercises
2nd Ed: P15.5, P15.6
(Hint: look at documentation for Scanner class!)
3rd Ed: P11.9, P11.11
4th Ed: P11.12, P11.14
Exceptions – Why do we need them?

• Remember the Account class? We added the following precondition to the deposit method:

\[ \text{amount} \geq 0 \]

• What if the client fails to check the precondition? The customers won't be happy to find out that sloppy programming has resulted in losing money because of a simple mistake!

Exceptions – Why we need them?

• Rather than using a precondition, we can have the method:

  • return a special value (e.g., true/false) to indicate whether or not the operation was successful

    \[ \text{not precise info about failure might not be checked} \]

  • print an error message

    \[ \text{who will see it?} \]

  • terminate the program

    \[ \text{over reaction} \]
Exceptions – Why we need them?

• Rather than using a precondition or one of the other methods suggested on the previous slide, we can have the method throw an exception if the amount is negative.

Benefits:
• We can force the client to acknowledge the problem.
• We allow the client to decide how to handle the problem.

What's a Java Exception?

• An exception is an object with a specific interface, that can be thrown.
• All exception classes are subclasses of the class Throwable defined in the Java library.
• Here are some of the methods of this class:
  Throwable();
  Throwable( String message );
  String getMessage();
  void printStackTrace();

• Exceptions encapsulate information about the kind of problem that has occurred (the message) and the sequence of method calls that led to the problem (the stack trace).
What's an exception?

- There are two types of exception: **checked** and **unchecked**.

- Unchecked exceptions are subclasses of Java’s `RuntimeException` class, while all others are checked exceptions.

- There is also an `Error` class that represents abnormal conditions that a program would normally not be expected to handle. Errors are treated like unchecked exceptions.

Java Exception Hierarchy

- Numerous exceptions and errors are defined in various java packages, i.e.,
  - `FileNotFoundException` in `java.io`
  - `IOException` in `java.io`
  - `NullPointerException` in `java.lang`
  - etc.

- Programmers can define their own exceptions as subclasses of `Exception` or its subclasses.
Defining an Exception Class

- Returning to our Account example, suppose we decide to throw an exception when the amount is negative.

- First we must decide which exception class to use. We could use the class Exception in the Java library but we can capture more information by defining our own exception class.

- Let's define a class named IllegalValueException to represent the type of exception that will be thrown when we attempt to pass a negative amount.

- This will be a checked exception (more about this later).

```java
public class IllegalValueException extends Exception {
    public IllegalValueException() {
    }
    public IllegalValueException(String msg) {
        super(msg);
    }
}
```
Throwing an Exception

/**
 * Deposit money into the account
 * @param amount The amount to be deposited
 * @pre true
 * @post IF amount >= 0
 * THEN getBalance() = @pre.getBalance() + amount
 * ELSE getBalance() = @pre.getBalance()
 * @throws IllegalValueException if amount is negative
 */
public void deposit(double amount) throws IllegalValueException {
    if (amount < 0)
        throw new IllegalValueException("Error: Neg. amount");
    balance = balance + amount;
}

Handling Exceptions

• Recall that IllegalValueException is a checked exception. This has consequences for a client calling our deposit method. The client code must do one of the following:

  • catch the exception

  • propagate (i.e., pass on) the exception to its caller (i.e., the method that called it)
Client Catching an Exception

public static void main( String[] args ) {
    Account instructorAccount =
        new Account ( "instructor", 100.0 );
    try {
        instructorAccount.deposit( 100 );
        System.out.println( "Balance: " +
            instructorAccount.getBalance() );
    }
    catch( IllegalValueException e ) {
        System.out.println( e.getMessage() );
    }
}

• What happens when deposit is called?
  nothing

What happens when this code executes?

public static void main( String[] args ) {
    Account instructorAccount =
        new Account ( "instructor", 100.0 );
    try {
        instructorAccount.deposit( -100 );
        System.out.println( "Balance: " +
            instructorAccount.getBalance() );
    }
    catch( IllegalValueException e ) {
        System.out.println( e.getMessage() );
    }
}

Error: Neg amount.
Client Propagating an Exception

```java
public void depositToAccount(Account anAccount,
   double amount) throws IllegalValueException
{
    anAccount.deposit(amount);
    System.out.println("Balance: "+
    anAccount.getBalance());
}
```

• The method that calls deposit must either:
  • catch the exception
  • propagate the exception to its caller
• If it propagates the exception then its caller must either catch or
  propagate the exception and so on …