# CSC 143 Java

**More on Exceptions**

*Reading: Ch. 15*



## Review

- Can throw exceptions to indicate unusual or erroneous conditions
  - Create an object to hold data about the kind of exception
  - Can be an instance of any subclass of Throwable
  - Throw it
- Can catch exceptions of desired kinds
- Exceptions help to separate the error handling code from the regular code



## Generic Exceptions

- So far, only throw exceptions of standard exception classes [such as??]

```
private void updateBalance( double amount) {
    if (this.balance + amount < 0) {
      throw new IllegalArgumentException("insufficient funds");
    } else {
      this.balance = this.balance + amount;
    }
  }
```

- To catch this error, must catch all IllegalArgumentExceptions... any drawbacks???



## Custom Exceptions

- A better way would be to create a special kind of exception, just for the error condition that's being reported

```
private void updateBalance( double amount) {
  if (this.balance + amount < 0) {
    throw new InsufficientFundsException( );
  } else {
    this.balance = this.balance + amount;
  }
}
```

- Think of two reasons why that's good...



## Custom Exception Classes

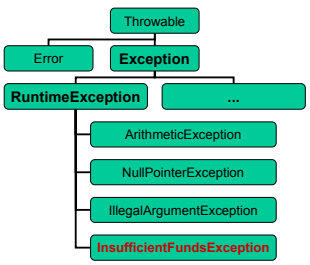
- To create a custom exception:

```
public class InsufficientFundsException extends RuntimeException {
    public InsufficientFundsException( ) {
        // invoke RuntimeException's constructor to set the error message
        super("insufficient funds");
    }
}
```

- Exception subclass is a regular class
  - may have instance variables, constructors, methods, etc. etc.



## Throwable/Exception Hierarchy



## Handling Custom Exceptions

- As expected:

```
try {
    mySavingsAccount.withdraw(100.00);
    myCheckingAccount.deposit(100.00);
} catch (InsufficientFundsException exn) {
    System.out.println("Transaction failed: " + exn.getMessage());
}
```

- Now won't accidentally catch unrelated IllegalArgumentExceptions



## Declaring Thrown Exceptions

- A method can declare the exceptions that it might throw, either directly or indirectly

```
public void withdraw(double amount) throws InsufficientFundsException {
  this.updateBalance(- amount);
}
private void updateBalance(double amount) throws InsufficientFundsException
{
  if (this.balance + amount < 0) {
    throw new InsufficientFundsException( );
  } else {
    this.balance = this.balance + amount;
  }
}
```



## *Throws* Clauses

- Syntax

  *<returnType> <messageName>(<arguments>)*
  **throws** *<ExceptionClass1>, ..., <ExceptionClassN>*
- For regular methods, abstract methods, methods in interfaces, and constructors
- Informs callers what exceptions they might need to handle
- It's good practice for a method to either handle all exceptions that may be raised by the messages it sends, or document using throws clauses that some exceptions might be propagated out of it to its callers



## *throw* vs. *throws*

- Two separate keywords in Java
- Make sure you know the difference!
  - Hint: where can you use each of them??

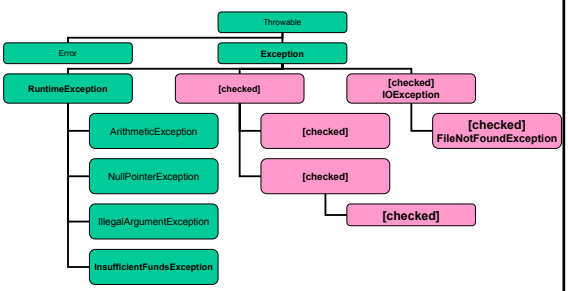


## Checked vs. Unchecked Exceptions

- For exception classes that are subclasses of RuntimeException, throws clauses are **optional**
  - The idea is that these exceptions might be so frequently thrown, that it would be pointless to list them everywhere
  - Called ***unchecked*** exceptions
- But for other exceptions, e.g. subclasses of Exception other than RuntimeException, throws clauses are **required**
  - Called ***checked*** exceptions



## Throwable/Exception Hierarchy



## Custom Checked Exceptions

- Java requires: all checked exceptions be handled, or be listed in the throws clause
- Good practice (according to some programmers): custom exceptions should be checked exceptions

```
public class InsufficientFundsException extends Exception { ... }

public void withdraw( double amount) throws InsufficientFundsException {…}
private void updateBalance( double amount)
    throws InsufficientFundsException { … }
public void transferTo(BankAccount other, double amount)
    // no throws clause needed, since all possible checked exceptions handled
{
    try { this.withdraw(amount);  other.deposit(amount);  }
    catch (InsufficientFundsException exn) { … }
}
```



## Throws Clauses and Overriding

- When one method overrides another
  - old rule: it must have the same signature (argument and result types)
  - new rule: it must have the same signature **and it must include the "same" exceptions in its throws clause ("same" since derived types are also OK)**
- One consequence: when declaring a general interface, e.g. for all Shapes, each method in the interface must mention all exceptions that any overriding method implementation might throw
  - This can be tricky to get right