APPinc.

Making Technology Reachable

Java Programming

RITA M. BARRIOS, Ph.D.

PRES. TRAINING & EDUCATION

Eclipse Shortcuts

- CTL-Shift-O Organize Imports
- CTL-Space Auto Complete
- CTL-Shift-F Format
- CTL-D Delete Current Line
- Alt-Shift-R Refactor Rename (variable, class, method)
- F3 Find Declaration
- F11 Run in Debug Mode

Source

Generate Getters and Setters

Generate Constructors

Preferences

-General

--Appearance

---Colors and Fonts

----Java Editor Text font

Exercise 1: Online Store Front – page 136

To get our heads around the interface concept, let's complete the following:

Create a Storefront application that uses packages, access controls, interfaces and encapsulation – this store manages the items in an online storefront to handle two main tasks

- Calculate the sales price of each item depending on how much is presently in stock
- Sort items according to sale price

Create two (2) classes, Storefront.java and Item.java in a new package called com.appinc.ecommerce

Exercise 1: continue

1. create a new package
   - Follow the instructions on page 131 – 133
2. Create the class Item.java on page 134 under the new package
3. Create the class Storefront on page 136 under the new package
4. Create the GiftShop application (has the main() method) on page 138 under the new package to run the storefront
5. Run GiftShop.java as a Java Application

Output:

```
Item ID: D01
Name: T SHIRT
Retail Price: $16.99
Price: $11.89
Quantity: 90

Item ID: C02
Name: LG MUG
Retail Price: $12.99
Price: $9.09
Quantity: 82

Item ID: C01
Name: MUG
Retail Price: $9.99
Price: $6.99
Quantity: 150

Item ID: C03
Name: MOUSEPAD
Retail Price: $10.49
Price: $5.25
Quantity: 800
```

Item.java

Line 1: establish the class is part of the package com.appinc.ecommerce

Line 3: implement the Comparable interface to make it easy to sort the class objects. It has hone method compareTo(Object) to return an integer – it compares two objects of a class

Lines 10-23: Item() constructor takes four String objects as arguments and uses them to set up the ID, name, retail price and quantity instance variables.

Lines 16-21: the value of the price instance variable is set depending on how much of that item is in stock

Line 22: rounds off rice

Lines 25-32: Simple accessor method

StoreFront.java

Line 6: Each product is an item object (item.java) and stored in LinkList instance variable (more on link lists later) called catalog

Line 8-13: creates a new object using the additem() method

Lines 15-17: getItem() is calles catalog.get(int) with an index as an argument returning the object stored at the location in the linked list

Lines 19-21: getItem() and getSize() are the interfaces to the information stored in a private catalog variable.

Lines 23-25: sort() use the implementation of the Comparable interface coded in the item.class – the class method Collections.sort() sorts the linked list

Java Debug Demo

- Setting Breakpoints
- Watching variables
- Stepping through a program
- Conditional Breakpoints
- Modifying Variables

What is an Interface?

- A named collection of method definitions without implementations
- An interface is similar to an abstract class with the following exceptions:
  - All methods defined in an interface are abstract. Interfaces can contain no implementation
  - Interfaces cannot contain instance variables. However, they can contain public static final variables (ie. constant class variables)
- Interfaces are declared using the "interface" keyword
  - If an interface is public, it must be contained in a file which has the same name.
- Interfaces are more abstract than abstract classes
- Interfaces are implemented by classes using the "implements" keyword.

Declaring an Interface

In Steerable.java:

```
public interface Steerable
{
  public void turnLeft(int degrees);
  public void turnRight(int degrees);
}
```

When a class "implements" an interface, the compiler ensures that it provides an implementation for all methods defined within the interface.

In Car.java:

```
 public class Car extends Vehicle implements Steerable
 {
   public int turnLeft(int degrees)
   {
         [...]
   }

   public int turnRight(int degrees)
   {
         [...]
   }
```

Exception Processing

CHAPTER 7

HANDLING EXCEPTIONS

try/catch/finally

Try catch finally

```
try {
    //do something
} catch (ExceptionType name) {

} catch (ExceptionType name) {

}  finally {
   //clean up
}
```

finally is always executed no matter if there an error or not.

Often used to close files

Checked Exceptions vs. Unchecked Exceptions

RuntimeException, Error and their subclasses are known as *unchecked exceptions*.

All other exceptions are known as *checked exceptions*, meaning that the compiler forces the programmer to check and deal with the exceptions.

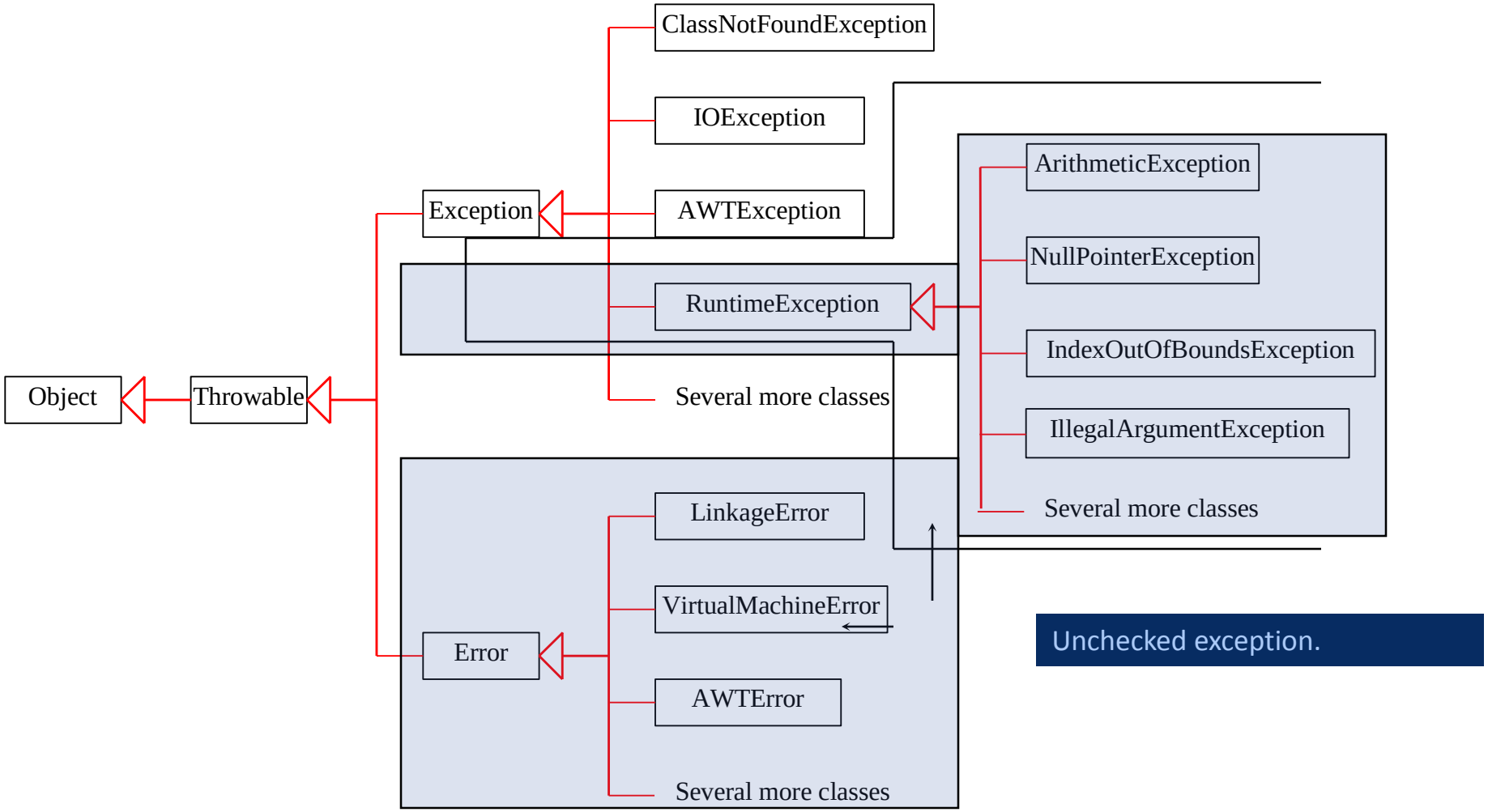
Unchecked Exceptions

In most cases, unchecked exceptions reflect programming logic errors that are not recoverable.

- **NullPointerException** is thrown if you access an object through a reference variable before an object is assigned to it
- **IndexOutOfBoundsException** is thrown if you access an element in an array outside the bounds of the array.

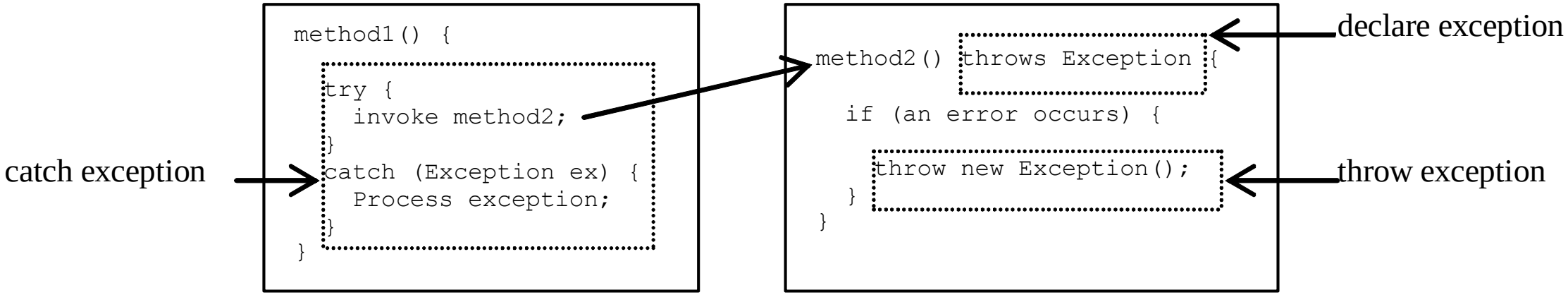
These are the logic errors that should be corrected in the program. Unchecked exceptions can occur anywhere in the program.

Checked or Unchecked Exceptions



Unchecked exception.

Declaring, Throwing, and Catching Exceptions



Throws – indicates that a method may throw an exception – method2 may throw an Exception and method1 catches it to process it

Declaring Exceptions

Every method must state the types of checked exceptions it might throw. This is known as *declaring exceptions*.

```
public void myMethod() throws IOException
```

```
public void myMethod() throws IOException, OtherException
```

Throwing Exceptions

When the program detects an error, the program can create an instance of an appropriate exception type and throw it.

This is known as *throwing an exception*. Here is an example,

```
throw new TheException();

TheException ex = new TheException();
throw ex;
```

Throwing Exceptions Example

```
/** Set a new radius */
 public void setRadius(double newRadius)
     throws IllegalArgumentException {
   if (newRadius >= 0)
     radius =  newRadius;
   else
     throw new IllegalArgumentException(
       "Radius cannot be negative");
 }
```

Catching Exceptions

```
try {
  statements;  // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Catch or Declare Checked Exceptions

Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must invoke it in a try-catch block or declare to throw the exception in the calling method.

method p1 invokes method p2 and p2 may throw a checked exception (e.g., IOException), you have to write the code as shown in (a) or (b).

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

The `finally` Clause – Always is executed

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

Trace a Program Execution

Suppose no exceptions in the statements

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

animation

Trace a Program Execution

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The final block is always executed

animation

Trace a Program Execution

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Next statement in the method is executed

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose an exception of type Exception1 is thrown in statement2

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The exception is handled.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The final block is always executed.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The next statement in the method is now executed.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Handling exception

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Execute the final block

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Rethrow the exception and control is transferred to the caller

animation

Cautions When Using Exceptions

Exception handling separates error-handling code from normal programming tasks, thus making programs easier to read and to modify.

Exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods.

When to Throw Exceptions

An exception occurs in a method.

If you want the exception to be processed by its caller, you should create an exception object and throw it.

If you can handle the exception in the method where it occurs, there is no need to throw it.

When to Use Exceptions

When should you use the try-catch block in the code?

You should use it to deal with unexpected error conditions.

Do not use it to deal with simple, expected situations. For example, the following code

```
try {
  System.out.println(refVar.toString());
}
catch (NullPointerException ex) {
  System.out.println("refVar is null");
}
```

When to Use Exceptions

is better to be replaced by

```
if (refVar != null)
  System.out.println(refVar.toString());
else
  System.out.println("refVar is null");
```

Creating Custom Exception Classes

- Use the exception classes in the API whenever possible.
- Create custom exception classes if the predefined classes are not sufficient.
- Declare custom exception classes by extending Exception or a subclass of Exception.

Exercise – Simple exception handling MusicTest1.java

In the following exercise, we use the javax.sound.midi.* library

To use the audio interface, we need to engage the speakers and tonal values of the computer system

We must access a “tool” called a sequencer to read the midi data and to send it to the right places

In order to determine what exceptions are thrown by a sequencer, we can look to the Oracle documentation

In the following exercise, we don’t actually play anything yet….

https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/MidiUnavailableException.html

javax.sound.midi

Class MidiUnavailableException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      javax.sound.midi.MidiUnavailableException

All Implemented Interfaces:

Serializable

---

public class **MidiUnavailableException**
extends Exception

A MidiUnavailableException is thrown when a requested MIDI component cannot be opened or created because it is unavailable. This often occurs when a device is in use by another application. More generally, it can occur when there is a finite number of a certain kind of resource that can be used for some purpose, and all of them are already in use (perhaps all by this application). For an example of the latter case, see the setReceiver method of Transmitter.

Exercise – MusicTest1.java

```java
package Exceptions;

// musical Instrument Digital Interface...

import javax.sound.midi.*;

// first get a sequencer object - takes in the midi data
// and sends it to the right place -- it plays the music

public class MusicTest1 {
    public void play() {

        // getSequencer() check the Oracle docs to determine what
        // exceptions it throws and use it in the catch
        //
        // here we are handling the exception

        try {
            Sequencer sequencer = MidiSystem.getSequencer();
            System.out.println("Successfully got a sequencer");
        } catch (MidiUnavailableException ex) {
            System.out.println("Didn't get one");
        }
    } // close play

    public static void main (String [] args){
        MusicTest1 mt = new MusicTest1();
        mt.play();
    } // close main
} // close class
```

Problems | Javadoc | Declaration | Console

```
<terminated> MusicTest1 [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 25, 2014, 10:16:14 PM)
Successfully got a sequencer
```

Exercise - Custom Exceptions

Sometimes we need to create custom exceptions based on the application requirements

We create our own exceptions by extending the Exception class

This is demonstrated in the following example:

Code the following and test as Java application when input is valid and when the exception is forced

- Uncomment line 6 then run
- Comment line 6, uncomment line 7 then run

```java
package Exceptions;

public class MyOwnException {
    public static void main(String[] a){
        try{
            //MyOwnException.myTest("this is a test"); // test not null string
            //MyOwnException.myTest(null);   // test null string w/ exception

        } catch(MyAppException mae){
            System.out.println("Inside catch block: "+mae.getMessage());
        }
    } // end of main

    static void myTest(String str) throws MyAppException{
        if(str == null){
            throw new MyAppException("String val is null");
        }else {
            System.out.println("All's good!");
        }
    } // end of myTest
} // end of class MyOwnException

class MyAppException extends Exception {

    private String message = null;
    public MyAppException() {
        super();
    }

    public MyAppException(String message) {
        super(message);
        this.message = message;
    }

    public MyAppException(Throwable cause) {
        super(cause);
    }

    @Override
    public String toString() {
        return message;
    }

    @Override
    public String getMessage() {
        return message;
    }
} // end of class MyAppException
```

```
<terminated> MyOwnException [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 25, 2014, 10:34:47 PM)
All's good!
```

Expand MusicTest1 to play something…

In this exercise we expand the original MusicTest1.java to actually play something

```java
package Exceptions;

import javax.sound.midi.*;

public class MiniMiniMusicApp {
    public static void main(String [] args){
        MiniMiniMusicApp mini = new MiniMiniMusicApp();
        mini.play();
    } // close main

    public void play(){
        try {
            Sequencer player = MidiSystem.getSequencer();
            player.open();

            Sequence seq = new Sequence(Sequence.PPQ, 4);
            Track track = seq.createTrack();

            //  music played as messages - set the instrument,
            //   set the message (music note), add it to the track, set the seq
            //   start the player

            //-- message 192 says change the instrument
            ShortMessage first = new ShortMessage();
            first.setMessage(192, 1, 102, 0); //- default is piano, 102 is sax
            MidiEvent changeInstrument = new MidiEvent(first,1);
            track.add(changeInstrument);

            // 144 = message type
            // 1 = channel - musician 1
            // 44 = note to play (0 - 127 low to high)
            // 100 = velocity (how hard and fast to press the key

            ShortMessage a = new ShortMessage();
            a.setMessage(144, 1, 44, 100);
            MidiEvent noteOn = new MidiEvent(a, 1); // duration
            track.add(noteOn);     // start playing

            ShortMessage b = new ShortMessage();
            b.setMessage(128, 1, 44, 100);
            MidiEvent noteOff = new MidiEvent(b, 16);
            track.add(noteOff);    // stop playing

            player.setSequence(seq);
            player.start();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    } // end play
} // class close
```

Basically building the tracks of a cd

Independent Exercise – Expand MiniMiniMusicApp.java

Use variables for instrument and note replacing the following

```
mini.play();  on line 8
```

With

```
mini.play(instrument, note);
```

Both variables should be declared as integers and have valid values of 0 – 127

Test with the following values

1 & 120 – Acoustic Grand Piano and high note of 120
11 & 90 – Music Box and lower note
14 & 80 – Xylophone and lower note

Solution – MiniMusic2.java

```java
package Exceptions;

import javax.sound.midi.*;

public class MiniMusic2 {
    public static void main (String [] args){
        MiniMusic2 mini = new MiniMusic2();

        int instrument = 1;  // Acoustic Grand Piano
        int note = 120;
        mini.play(instrument, note);

        int instrument2 = 11; // Music Box
        int note2 = 90;
        mini.play(instrument2, note2);

        int instrument3 = 14; // xylophone
        int note3 = 80;
        mini.play(instrument3, note3);

    }// end of main

    public void play(int instrument, int note){
        try {
            Sequencer player = MidiSystem.getSequencer();
            player.open();
            Sequence seq = new Sequence(Sequence.PPQ, 4);
            Track track = seq.createTrack();

            MidiEvent event = null;

            ShortMessage first = new ShortMessage();
            first.setMessage(192, 1, instrument, 0);
            MidiEvent changeInstrument = new MidiEvent(first,1);
            track.add(changeInstrument);

            ShortMessage a = new ShortMessage();
            a.setMessage(144, 1, note, 100);
            MidiEvent noteOn = new MidiEvent(a, 1);
            track.add(noteOn);

            ShortMessage b = new ShortMessage();
            b.setMessage(128, 1, note, 100);
            MidiEvent noteOff = new MidiEvent(b, 16);
            track.add(noteOff);

            player.setSequence(seq);
            player.start();

        } catch (Exception ex) {ex.printStackTrace();}
    }

}  // end of MiniMusic2
```

Questions?

Open Discussion

A basic process to writing code

- ✓ Figure out what the class is suppose to do
- ✓ List the instance variables and methods
- ✓ Write pseudo code for the methods
- ✓ Write test code for the methods
- ✓ Implement the class
- ✓ Test the methods
- ✓ Debug and implement as needed

A Simple Battleship like game: SimpleDotCom Game

In this game, we kill off dot coms vs battleships in as fewest number of guesses as possible. We'll continue to build it as we go and it will take several refactoring iterations to accomplish all of its goals

- Primary Objective: When the program is launched, it will launch on a virtual 7 x 7 grid
- We haven't learned about GUIs yet so this version will use the command line
- We will get a response of hit, miss or You sunk Pets.com or whatever dot com is sunk
- The game ends when three dot coms have been sunk and you will see your rating
- Each dot com takes up three cells

A							
B							
C							
D							
E							
F							
G							
	0	1	2	3	4	5	6

Go2.com

Pets.com

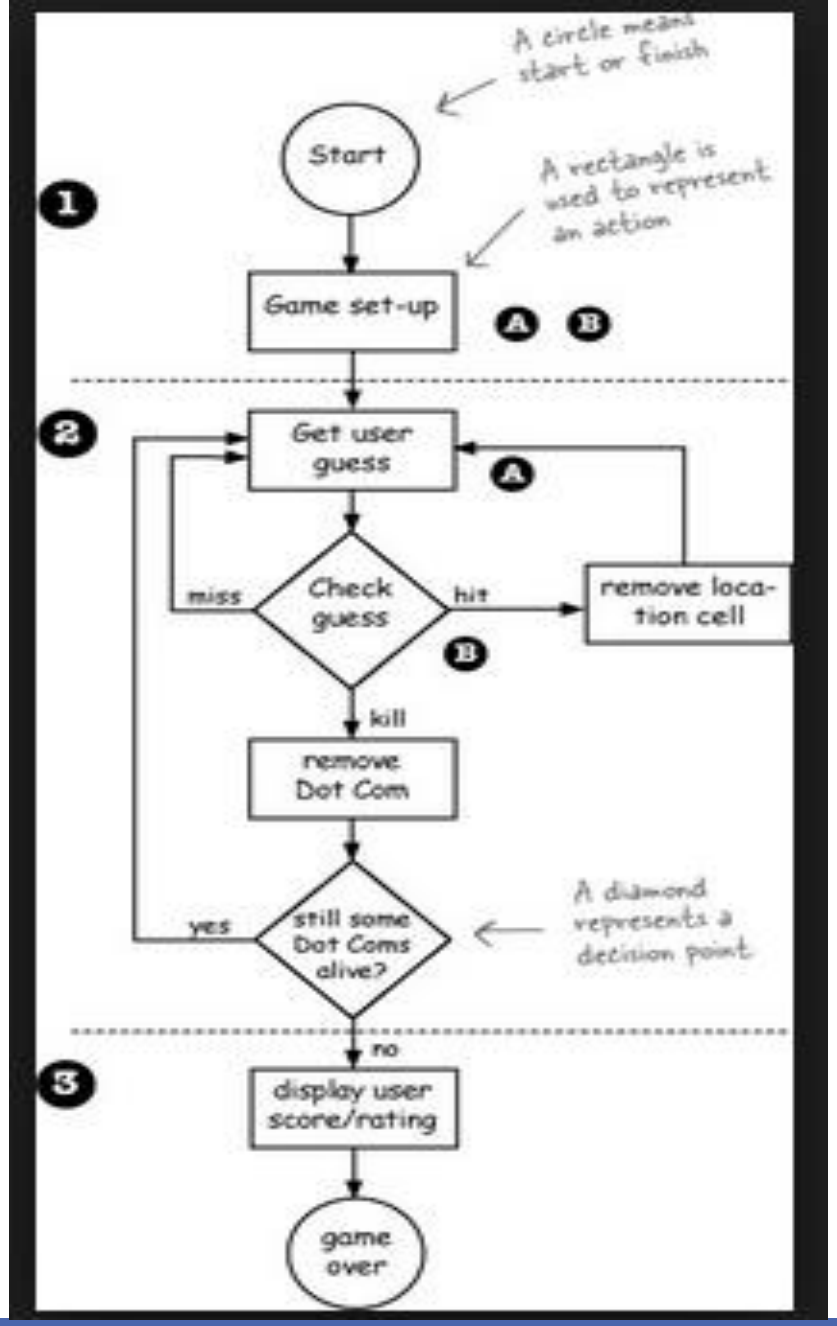
AskMe.com

7 x 7 grid – virtual game board

High-Level Design

We need Classes and Methods **Figure out the general flow of the game**

1. **User Starts the game**
   A. Game create 3 dot coms
   B. Game places 3 dot coms on the virtual grid
2. **Game play begins** – repeat the following until there are no more dot coms
   A. Prompt the user for a guess (eventually…)
   B. Check the user's guess to look for a hit, miss or kill – Take appropriate action:
      a. If hit – delete a cell
      b. If a kill – delete the dot com
3. **Game finishes**
   A. Give user a rating



High-Level Design (2)

Figure out what kind of objects we need

- Focus on things – What do we need?
- We'll need at least 2 classes – a game class and a DotCom class

Let's start with a simple version

- 1 dimension array
- 1 dot com
- No instance variables
- The whole game is coded in main()

Write the Pseudo Code

What variables need to be declared?

What methods do we need? What actions does the Business Requirements tell us that must happen?

What are the attributes and behavior of the methods? Are they setters or getters?

Pseudo Code – Declare Statements

Declare an int array to hold the location cells

Declare an int to hold the number of hits – set it to 0

Declare a checkYourself() method that takes a String for the guess, checks it and returns the result representing hit, miss or kill

Delcare a setLocationCells() setter method that takes an int array which has 3 cell locations

Pseudo Code – Method checkYourself

```
Method String checkYourself(String userGuess)
  Get the user guess as a String parameter
  Convert the user guess into an int using Integer.parseInt(stringGuess)
  Repeat with each location in the int array
     // compare the user guess with the location cell
     if the user guess matches
          increment the number of hits
          if number of hits = 3, return kill
          else return hit
          end if
    else return miss
    end if
  end repeat
End method
```

Psuedo Code – setLocation Method

```
Method void setLocationCells(int [] cellLocations)
  get the cell locations as an int array parameter
  assign the cell location parameter to cell location inatance
End method
```

Let’s Code 2 Modules

Using the DotComGame.pdf on google group, complete the following:

1. Create a SimpleDotCom.java class based on the requirements

2. Create a SimpleDotComTestDrive.java file to test the class

```
Problems  @ Javadoc  Declaration  Console
<terminated> SimpleDotComTestDrive [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 16, 2014,
hit
hit
kill
```

SimpleDotCom Game

SimpleDotComTestDrive.java | SimpleDotCom.java

```java
public class SimpleDotComTestDrive {
    public static void main (String [] args){
        SimpleDotCom dot = new SimpleDotCom();
        int [] locations = {2, 3, 4};
        dot.setLocationCells(locations);
        String userGuess = "2";
        String result = dot.checkYourself(userGuess);

        String userGuess2 = "9";
        String result2 = dot.checkYourself(userGuess);

        String userGuess3 = "4";
        String result3 = dot.checkYourself(userGuess);


    }
}
```

Problems | Javadoc | Declaration | Console

```
<terminated> SimpleDotComTestDrive [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (Nov 16, 2014,
hit
hit
kill
```

```java
public class SimpleDotCom {
    int[] locationCells;
    int numOfHits = 0;

    public void setLocationCells(int [] locs) {
        locationCells = locs;
    }

    public String checkYourself(String stringGuess) {
        int guess = Integer.parseInt(stringGuess);
        String result = "miss";
        for (int cell : locationCells){
            if (guess == cell) {
                result = "hit";
                numOfHits++;
                break;
            }
        } // out of the loop

        if (numOfHits == locationCells.length) {
            result = "kill";
        }

        System.out.println(result);
        return result;
    } // close method
} // close class
```

Let’s Examine the Code in our game

checkYourself()

Line 10: converts the string to an integer using Integer.parseInt()

Line 12: for (int cell : locationCells) {} – this is an enhanced for loop that began with Java 5 – can use the original format too

The colon : means “in” so the statement means “for each int value in the array locationCells – This new format of the for loop iterates over an array while looping.

Each time thru the loop, the next element in the array will be assigned to the variable cell (holds only one element of the array)

Continuing on with DotCom Game….

So far, we've created the SimpleDotCom class and a test drive program.

We need to create the real game program

SimpleDotCom Game – Part 2

STEP 1 – Create the Pseudo Code for SimpleDotComGame class

Referring back to the code you created for SimpleDotCom.java and the software requirements, create the game's pseudo code

You've been given a few lines to start…. Page 1

SimpleDotCom Game – Pseudo Code Solution – Page 2

public static void main (String [] argas)

DECLARE an int variable to hold the number of user guesses, name it numOfGuesses and set it to 0

MAKE a new SimpleDotCom instance

COMPUTE a random number between 0 and 4 that will be the starting location cell position

MAKE an int array with 3 ints using the randomly generated number; the starting random number is generated by 1, then by 2 (for example, if we generate 1, then incrementing it by 1 will result in 2 and then incrementing it by 2 will result in 3 = 1, 2, 3

INVOKE the setLocationCells() method on the SimpleDotCom instance

DECLARE a boolean variable representing the state of the game – named isAlive – SET it to true

WHILE the dot com is still alive (isAlive == true):

GET user input from the command line

// **CHECK** the user guess

INVOKE the checkYourself() method on the SimpleDotCom instance

INCREMENT numOfGuesses variable

// **CHECK** for dot com death

IF result is "kill"

SET isAlive to false (don't enter the loop again)

PRINT the number of user guesses

END IF

END WHILE

END METHOD

SimpleDotComGame – Real Code

New Statements

- Math.random()
  - Java class Math with random () method that generates random numbers

```
int randomNum = (int) (Math.random() * 5);
```

- getUserInput()
  - A helper class that we've given you to accept user input from the command line (console in Eclipse)

```
String guess = helper.getUserInput("enter a number");
```

SimpleDotCom Game – Part 2

STEP 2 – Create the Real Code for SimpleDotComGame class

1st – code the helper class GameHelper.java exactly as shown on page three – it uses new features that we will talk about later as to not muddy the process of problem solving and translating pseudo code to real code

2nd – code your SimpleDotComGame.java file using your pseudo code – test as a Java Application by enter user input into the console

SimpleDotComGame.java Solution – Page 4

```java
package SimpleDotComGame;

class SimpleDotComGame {
    public static void main (String [] args){
        int numOfGuesses = 0;
        GameHelper helper = new GameHelper();

        SimpleDotCom theDotCom = new SimpleDotCom();
        int randomNum = (int) (Math.random() * 5);
        System.out.println(randomNum);

        int[] locations = {randomNum, randomNum + 1, randomNum + 2};
        theDotCom.setLocationCells(locations);
        boolean isAlive = true;

        while (isAlive == true) {
            String guess = helper.getUserInput("enter a number");
            String result = theDotCom.checkYourself(guess);
            numOfGuesses++;
            if (result.equals("kill")){
                isAlive = false;
                System.out.println("You took " + numOfGuesses + " guesses");
            } // close if
        } // close while
    } // close main
} // end class
```

Problems | @ Javadoc | Declaration | Console

```
<terminated> SimpleDotComGame [Java Application]
4
enter a number 4
hit
enter a number 7
miss
enter a number 5
hit
enter a number 8
miss
enter a number 6
kill
You took 5 guesses
```