3134 Data Structures in Java

Lecture 2
Jan 22 2007
Shlomo Hershkop
Announcements

- **Ta office hours:**
  - Jonathan DePeri
    - Wed 4:15-6pm
  - Matt Chu
    - Thur 4-6pm
  - Sara Stolbach
    - Mon 4:15-5:15, Thu 2:30-3:30
Review Basics

- I am going to start a review of Java

- Want to schedule more sessions (Tas)
  - Want to know what works
  - Will be hands on based + review

- By the end of the week you should be familiar enough with basic java programming to be able to work on the homework

- Need to practice if this is really new to you, and please stop by to talk
Java Refresher Course

3134 – DS in Java
Spring 2007
Basics I
First Steps

- **Java Class**
  - Creating a class
  - Class primitive members
  - Constructors
  - Compile/run

- **Main function**
  - Adding main
  - Importing classes
  - Instantiation
  - Compile/run
Word of caution

- Please don’t adopt a “copy-paste” approach to programming
- Understand what you are doing
- Feel free to explore
  - It’s fun! Trust us
Errors

- It is normal to generate errors

- Each type of error needs a different approach
Compile time errors

- These are errors generate when the program tries to compile

- In eclipse will become colorful red lines
  - Can mouse over to get an idea

- Outside of eclipse, if call the javac.exe compiler will see warnings with line numbers
Run time errors

- There are errors generated by program logic

- Pay attention to run time errors
  - What class involved
  - What line (debuggers)
  - What method involved
  - Sequence trace
Logic Errors

- A program may run, but produce incorrect results, perhaps using an incorrect formula (logical errors)

- Look at your results of your output to make sure it makes sense (if applicable)
API Documents

- Unlike other languages, Java has many libraries bundled by default.

- Application Programming Interface (API) docs, are the view given to the programmer:
  - What classes are out there
  - How to invoke and use them

- Please don’t reinvent the wheel if it exists already (unless specified).
  Example:
  - java.lang.String
  - java.util.StringTokenizer
method

- A method is simply a bunch of instructions which you can call using a shorter name

- Don’t call it foo()

- Example:
  - getProfessorName
    - Pretty obvious what this does
    - But what are we working on?
The basic object in the Java language is a *class*

A class is the blueprint of how you can describe an object

The class uses methods to define the behaviors of the object

Example:

If you want to represent a person methods would include:

- `getDateOfBirth`
- `GetHairColor`
- `GetName`
- A class represents a concept, and a Java object represents the embodiment of that concept.

- For example:
  - Person Class represents the idea of a person.

- Multiple objects can be created from the same class:
  - Person A, B, C

- We have 3 instances of the person class which we are calling A, B, and C.
Special method

- You can have as many classes as you like
- One (at least) will contain a special method called “Main”
- The class that contains the main method of a Java program represents the starting point in your program
- Code will start running from there
Java packages

- Collection of similar classes
- Package names are dot-separated identifier sequences

java.util
javax.swing
com.sun.misc
edu.columbia.cs.robotics
Packages

- Unique package names: start with reverse domain name
- Corresponds to directory structure
  - Must match directory structure
- package statement to top of file
- Class without package name is in "default package"
- Full name of class = package name + class name
  
  `java.util.String`
Importing Packages

- Tedious to use full class names
- import allows you to use short class name

```java
import java.util.Scanner;

Scanner a;

Instead of saying: java.util.Scanner a
```

- Can import all classes from a package using star
```java
import java.util.*;
```
public class Test {
    /* A comment */
    public static void main(String[] args) {
        
    }
}
Constructing

- Once we define a class we create an instance of the class.
- The constructor method is responsible for initializing the object.
- The constructor is the method with the same name as your class.
- command new creates an instance

Person A = new Person();
Objects and Classes

A class
(the concept)

Bank Account

An object
(the realization)

John’s Bank Account
Balance: $5,257

Bill’s Bank Account
Balance: $1,245,069

Mary’s Bank Account
Balance: $16,833

Multiple objects
from the same class
String is a bundled class

- String name = new String(“Prof”);
- Let's look at the String API
  - name.length();
  - String two = name.sub(0,1);
Problem

- String s;
  - s.length;

- What is wrong here?
Next

- Primitive types
- Primitive arrays
- Code Blocks
- Control of Code
Primitive types

- byte
- short
- int
- long
- double
- float
- boolean
- char

- Byte
- Short
- Integer
- Long
- Double
- Float
- Boolean
- Character
Example

- int a = 10;
- a = 13 * a;
- a++;

- float averageRun = 23 / 2;
Arrays

- Can string together a fixed number of the same type in java by using an array to hold them together

- `int ages[] = int[10];`

- Will give you access to individual elements by indexing in the array
  - `ages[0] to ages[9] to store 10 numbers in memory`
First index

Element (at index 8)

Array length is 10
Block of code

- Curly braces are used to denote area of code
- Functions
- Something() {
  
}  
- Can freely use it anywhere you want
- All commands inside a block executed one after another
Scope

- Scope refers to where java programming objects variables/methods/classes can be accessed.
- Local
- Global
- Package
- Universal
Variables

- Variables declared within a method are local to that method
  - Local scope
- Variables declared within a class, are called field variables
  - Class wide scope
    - Including subclasses
    - Package wide scope
- Local variable can have the same name as field variables
  - Use this to disambiguate
Example

- `int x = 10;
- {
  - `int x = 5;
  - `x++;
  - `//what is x here ?
- }
- `//what is x here ?
Variable modifiers

- When variables exist as part of a class
- Can modify their permission

- Public
- Private
- Static
Example

```java
public class Person{
    public String name;
    private Date dateOfBirth;
    private String socialSecNumber;
}
```
Methods

- Methods are useful blocks of code
- isLegalDriver()
- hasMoney()
Types of methods

- **Accessor**
  - get some state information from the object

- **Mutator**
  - change information

- **Helper**
  - internal functions to accomplish tasks

- **Predicate**
  - help answer simple yes/no questions
Methods

1. Modifier
2. What we are returning
3. Method name
4. What variables you are taking in
   1. Parameter list
5. Exception list
6. {   } actual code
Method signature

- The signature is what uniquely identifies a method

- Method Overloading
Overloading

- Have the same method take different sets of arguments

- Example

- Constructor
  - Person(String)
  - Person(String, Date)
  - Person(String, Date, int)
Exceptions

- Object that represents an unusual event or an error
- Attempt to divide by zero
- Array out of bounds
- Null reference
Exceptions

- Tool to handle error during program run
  - Exception == exceptional event

- Idea: when an error occurs, a method can create an Object representing the error and hand it to the run time system

- The runtime system now tries to find someone to handle the particular error, it uses the call stack to find a handler
Exception handlers

- Are defined by your catch expression
- If a specific method doesn’t know how to handle the specific exception, it forwards it up the stack
- Remember: can have multiple catch blocks one after other
  - Exceptions have a hierarchy, they will be evaluated from highest to lowest, so the catch blocks must be in reverse order.
The birth of an exception

- You might use a method which might throw an exception
- You might create a method which creates and exception
- Your code might trigger an exception
Simple Example

```java
try {
    // you will do something which
    // might trigger an exception
}
catch(ExceptionType name) {
    // deal with it
}
```
Your method

public boolean checkBalance(int account) throws InvalidAccountException{

    if(account==null || account < 1){
        throw new InvalidAccountException("Bad Account Number");
    }

    ...
    ...
}

Chaining Exceptions

```java
try {
...
} catch (IOException e) {
    throw new SampleException("Other IOException", e);
}
```
Point

- Can deal with the problem
  - Ask user for help
  - Figure out what should be done
  - Log the error
  - Print a trace to debug
  - Die (ARGHUUUUUUUU!)
Exception Handling

- Example: NullPointerException

```java
String name = null;
int n = name.length(); // ERROR
```

- Cannot apply a method to null reference
- Virtual machine throws exception
- Unless there is a handler, program exits with stack trace

```
Exception in thread "main" java.lang.NullPointerException
at Student.setname(Student.java:15)
at StudentTest.main(StudentTest.java:20)
```
Checked and Unchecked Exceptions

- Compiler tracks only checked exceptions
- NullPointerException is not checked
- IOException is checked
- Generally, checked exceptions are thrown for reasons beyond the programmer's control
- Two approaches for dealing with checked exceptions
  - Declare the exception in the method header (preferred)
  - Catch the exception
Declaring Checked Exceptions

- Example, let's look at API for opening a file
- It may throw `FileNotFoundException`
- We choose not to deal with it as follows:

```java
public void read(String filename) throws FileNotFoundException {
    FileReader reader = new FileReader(filename);
    ...}
```

- Can declare multiple exceptions

```java
public void read(String filename) throws IOException, ClassNotFoundException {
    FileReader reader = new FileReader(filename);
    ...}
```

```java
public static void main(String[] args) throws IOException, ClassNotFoundException {
    ...
}
```
Catching Exceptions

try
{
    code that might throw an IOException
}
catch (IOException exception)
{
    take corrective action
}

- Corrective action can be:
  - Notify user of error and offer to read another file
  - Log error in error report file
  - In student programs: print stack trace and exit

exception.printStackTrace();
System.exit(1);
The **finally** Clause

- Will ALWAYS execute code block
  - Even if return statement in try block
- Cleanup needs to occur during normal and exceptional processing
- Example: Close a file

```java
FileReader reader = null;
try {
    reader = new FileReader(name);
    ...
} catch(...) {
    finally {
        if (reader != null) reader.close();
    }
}
```
null

- **null refers to no object**
  - Uninitialized objects
  - Explicit assignment
- **Can assign null to object variable:**
  - `Person p = null;`
- **Can test whether reference is null**
  - `if (p == null) ...`
- **Dereferencing null causes**
  `NullPointerException`
**this**

- Refers to own class (yourself)

```java
public boolean equals(Person other) {
    if (this == other) {
        return true;
    }
    return name.equals(other.name) && getBirthday.equals(other.getBirthDay());
}
```
Command Line Arguments

public static void main(String[] args)

- `args`, is an array of string.
- The elements of `args` are the command line arguments using in running this class.

Java testProgram -t -Moo=boo out.txt
0: `-t`
1: `-Moo=boo`
2: `out.txt`