CS 5010: Programming Design Paradigms
Fall 2017

Lecture 2: Whirlwind Tour of Java

Tamara Bonaci
t.bonaci@northeastern.edu

Administrivia 1

- First assignment due on Monday, Sept 18 by 6pm

- Code walkthroughs on Tuesday, Sept 19 from 10am-10pm, in 401 Terry Ave N:
  - 10am – 5:30pm in Camano classroom
  - 6pm – 10pm in Cypress classroom
  - → You should sign up for your walkthrough slot using this Google doc

Administrivia 2

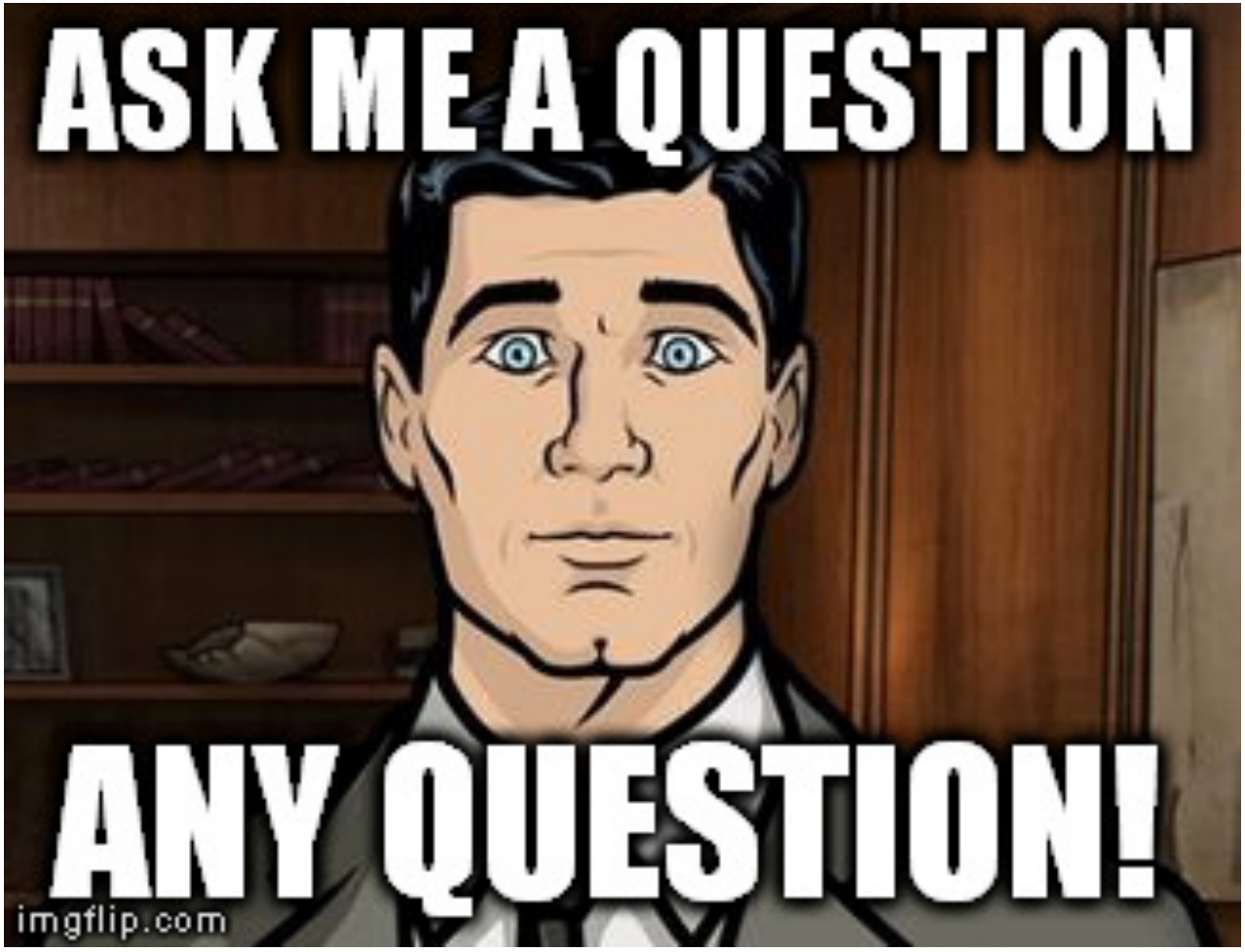
- How to submit your homework? Using CCIS GitHub
- By now, you should have:
  1. Setup your CCIS account
  2. Add your CCIS account to this Google doc
  3. Access your CCIS GitHub at least once
- If you haven't done this, or you don't have access to your GitHub repo, email Divya Agarwal ASAP (agarwal.d@husky.neu.edu)

Administrivia 3

- What To Do If I Have Questions?
  1. General question – post it publicly on Piazza
  2. Assignment specific question/sensitive question
     - Send us an email/private Piazza message, or
     - Come to office hours

- Office Hours:
  - Maria – Mondays 2:30-5:30pm in Lummi
  - Tamara – Tuesdays 10:00-12:00pm in Lummi
  - Tamara – this Thursday 2:00-4:00pm in Lummi
  - Adrienne – this Friday 4:00-5:00pm in Lummi

Your Questions



[Meme credit: imgflip.com]

Agenda – Whirlwind Tour of Java

- Introduction to Java
- Objects and classes in Java
- Data Types in Java
- Modifier Types in Java
- Scanner, String and Random Classes
- Assertions and Exceptions
- Testing with Java
- Javadoc
- HashCode
- Overview of Java 8 and 9

Whirlwind Tour of Java

INTRODUCTION TO JAVA

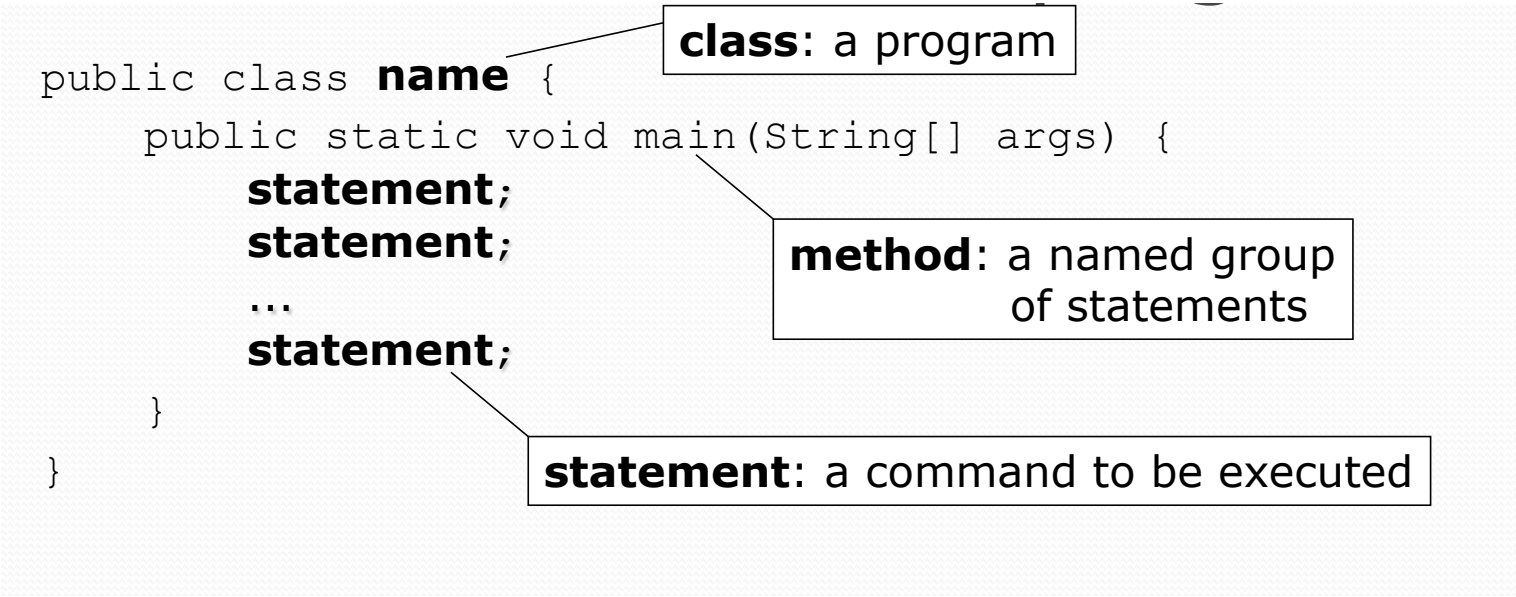
Java – The Most Popular Programming Language in the World*

- Object-oriented
- Many well-developed IDEs
- Tons of pre-written software (not all of it good ☺)
- Platform independent
  - Java compiler compiles the source code into byte code which runs on many different computer types

* https://www.tiobe.com/tiobe-index//

Structure of a Java Program

- Every executable Java program consists of a class,
  - That contains a method named `main`
    - That contains statements (commands) to be executed



[Graphic credit: Paerson Education]

Our First Java Program

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, PDP_Fall_2017!");
        System.out.println();
        System.out.println("Keep up the good work");
    }
}
```

- Output???
- System.out.println – two modes
  - Print a message: System.out.println("Hi!");
  - Print a blank line: System.out.println();

Static Methods

- Procedural decomposition – division of a problem into smaller units (methods)

- Static method - a named group of statements
  - Denotes the structure of a program
  - Eliminates redundancy through code reuse

- Steps to using a static method:
  - Declare it
  - Run in

Whirlwind Tour of Java

OBJECTS AND CLASSES IN JAVA

Objects and Classes

- Object – an entity consisting of states and behavior
  - States stored in variables/fields
  - Behavior represented through methods

- Classes – templates/blueprints describing the states and behavior that an object of that type supports

Classes in Java

- Classes – templates/blueprints describing the states and behavior that an object of that type supports
- Classes contain:
  - Local variables – variables defined within any method, constructor or block
    - *These variables are destroyed when the method has completed*
  - Instance variables – variables within a class, but outside any method
    - *Can be accessed from inside any method, constructor or blocks of that particular class*
  - Class variables – variables declared within a class, outside of any method, with the keyword **static**

Classes and Constructors in Java

- Classes – templates/blueprints describing the states and behavior that an object of that type supports

- Every class has a **constructor**
  - *In Java, if we don't explicitly write a constructor, Java compiler builds a default constructor for that class*
  - *But it is a good practice to write a constructor, even an empty one*

Classes and Constructors in Java

- Example: constructors for a class Zoo

```
public class Zoo{
    public Zoo() {
    }

    public Zoo(String name, String city, String state{
        // This constructor has three parameters,
        name, city and state.
    }
}
```

Creating an Object in Java

- In Java, an object is created from a class using a keyword **`new`**

- Three steps involved when creating an object from a class:
  1. Declaration – a new variable is declared, with a variable name, and object type
  2. Instantiation – an object is created using the keyword **`new`**
  3. Initialization – an object is initialized using the keyword **`new`** + a constructor call

Creating an Object in Java

- Example: creating an object Zoo

```
public class Zoo{
    // This constructor has three parameters, name, city
    and state.
    public Zoo(String name, String city, String state) {
        System.out.println("Passed Name is :" + name );
    }
    public static void main(String []args) {
    // Following statement would create an object myZoo
    Zoo myZoo = new Zoo( “Woodland Park”, “Seattle”, “WA” );
    }
}
```

Accessing Instance Variables and Methods

- In Java, instance variables and methods are accessed via created objects:

```
/*First create an object */
ObjectReference = new Constructor();

/*Now call a variable as follows */
ObjectReference.variableName;

/*Now call a variable as follows */
ObjectReference.methodName();
```

Accessing Instance Variables and Methods

```
public class Zoo{
        float ZooSize;

        public Zoo(String name) {
                // This constructor has one parameter, name.
                System.out.println("Zoo's name is :" + name );
        }
        public void setSize(float size) {
                ZooSize = size; }

        public float getSize( ) {
                return ZooSize; }

        public static void main(String []args) {
                Zoo myZoo = new Zoo( "Woodland Park" );

                /* Call class method to set zoo size*/
                myZoo.setSize(55 );

                /* Call another class method to get zoo size */
                myZoo.getSize( );
        }
}
```

Whirlwind Tour of Java

DATA TYPES IN JAVA

Data Types in Java

- Data type - a category or set of data values
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types

- Java distinguishes between
  - Primitive data types – store the actual values
  - Reference data types – store addresses to objects that they refer to

Primitive Data Types in Java

- Eight primitive data types are supported in Java:
  - **byte** – 8-bit signed two's complement integer (min. value -128, max value 127)
  - **short** – 16-bit signed two's complement integer (min. value -32,768, max. value 32,767)
  - **int** – 32-bit signed two's complement integer (min. value -2^{31}, max. value 2^{31} -1)
  - **long** – 64-bit two's complelent integer (min. value -2^{63}, max. value 2^{63}-1)
  - **float** – single-precision 32-bit IEEE 754 floating point
  - **double** – double-precision 64-bit IEEE 754 floating point
  - **boolean** – only two possible values, true and false
  - **char** – single 16-bit Unicode character (min. value '\u0000' (0), max. value '\uffff' (65, 535))

Primitive Data Types in Java

- Eight primitive data types are supported in Java:
  - **byte**, **short**, **int**, **long**, **float**, **double, boolean, char**

- Java also provides special support for character strings via **java.lang.String class**

Reference Data Types in Java

In Java, reference data types are:

- Created using classes' constructors
- Used to access object of a declared type, or any object of a *compatible* type

- Some examples of reference data types:
  - String
  - Scanner
  - Random

Data Types in Java - Summary

- Question: what happens if we declare a variable, but don't initialize it?
- Answer: Most likely, the Java compiler will set those variables to reasonable default values

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

But don't do that → not initializing your data generally considered a bad programming style

Whirlwind Tour of Java

MODIFIER TYPES IN JAVA

Modifiers in Java

- Modifiers – keywords *preceding* the rest of the statement, used to change the meaning of the definitions of a class, method, or a variable

- Modifiers in Java can be:
  - Access control modifiers
  - Non-access control modifiers

Access-Control Modifiers in Java

- In Java, there exist four access levels:
  1. Visible to the package (default, no modifier needed)
  2. Visible to the class only (modifier `private`)
  3. Visible to the world (modifier `public`)
  4. Visible to the package and all subclasses (modifier `protected`)

Non Access-Control Modifiers in Java

- Non-access control modifiers in Java include:
  - `static` - for creating class methods and variables
  - `final` – for finalizing the implementations of classes, methods and variables
  - `abstract` – for creating abstract classes and methods
  - `synchronized` and `volatile` – used for threads

Whirlwind Tour of Java

SCANNER, STRING AND RANDOM CLASSES

Interactive Programs

- Interactive programs – programs where a user interacts with a program by providing an input into the console, that a program then reads and uses for execution

- Interactive programs can (sometimes) be challenging
  - Computers and users think in very different way
  - Users misbehave
  - Users are malicious

Interactive Programs in Java

- Interactive programs – programs where a user interacts with a program by providing an input into the console, that a program then reads and uses for execution

- Java typically handles user input using System.in, but
  - System.in is not intended to be used directly
  - Instead, we use a second object, `Scanner`
  - Scanner is in a package named java.util

- Constructing a Scanner object to read console input:

```
import java.util.*;
Scanner name = new Scanner(System.in);
```

Scanner Methods

Method	Description
nextInt()	Reads a token of user input as an int
nextDouble()	Reads a token of user input as a double
next()	Reads a token of user input as a String
nextLine()	Reads a line of user input as a String

[Table credit: Paerson Education]

- Each method waits until the user presses *Enter*
  - The value typed is returned

Example Scanner Usage

```
import java.util.*; // so that we can use Scanner

public class ReadSomeInput {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("How many courses are you taking this term? ");
        int numCourses = console.nextInt();

        System.out.println(numCourses + "... That's too many!");
    }
}
```

Output (user input underlined):

How many courses are you taking this term? 4
4... That's too many!

Strings in Java

- String - an object storing a sequence of text characters
  - *Unlike most other objects, a String object is not created with new**
    - String name = "text";
    - String name = expression;
- Example

```
int x = 5;
int y = 25;
String point = "(" + x + ", " + y + ")";
```

* This is not the whole story. `String` can be created as an object, with a keyword **`new`**. To learn more about why are `Strings` special in Java, please refer to Java Programming Tutorial, Java String is Special, [Online], https://www.ntu.edu.sg/home/ehchua/programming/java/J3d_String.html

Strings Indexes

- Characters of a string are numbered with 0-based indexes:
- Example:

```
String name = "NEU CCIS";
```

Index	0	1	2	3	4	5	6	7
Char	N	E	U		C	C	I	S

- The first character's index is always 0
- The last character's index is 1 less than the string's length
- The individual characters are values of type char

Strings Methods

- String methods are called using *dot notation*
- Example:
  - String city = "Seattle";
  - System.out.println(city.length()); //7

Method name	Description
indexOf(str)	Index where the start of the given string appears in this string (-1 if it is not there)
length()	Number of characters in the string
substring(index1, index 2), substring(index1)	The characters in this string from *index1* (inclusive) to *index2* (exclusive). If *index2* omitted, grab till the end of string.
toLowerCase()	A new string with all lowercase letters
toUpperCase()	A new string with all uppercase letters

[Table credit: Paerson Education]

Modifying Strings

- Some methods (e.g., substring, toLowerCase, etc) create/return a new string, rather than modifying the current string

```
String s = ”northeastern";
s.toUpperCase();
System.out.println(s); // northeastern
```

- To modify a variable, you must reassign it:

```
String s = ”northeastern";
s = s.toUpperCase();
System.out.println(s); // NORTHEASTERN
```

Strings as User Inputs

- Scanner's **next** method reads a word of input as a String

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
   " letters and starts with " + name.substring(0, 1));
```

- Output 1:

```
What is your name? Tamara
TAMARA has 6 letters and starts with T
```

- Output 2:
  - What is your name? Mary Ann
  - MARY ANN has 8 letters and starts with M
- The nextLine method reads a line of input as a String

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

Comparing Strings

- Relational operators such as < and == fail on objects

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();

if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- This code will compile, but it will not print the song. Why?
  - == compares objects by references, so it often gives **false** even when two Strings have the same letters

The equals() Method

- Objects are compared using a method named **equals**

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();

if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technically, this is a method that returns a value of type boolean, the type used in logical tests

String Test Methods

Method	Description
equals(str)	Whether two string contain the same characters
equalsIgnoreCase(str)	Whether two strings contain the same characters, ignoring upper vs. lower case
startsWith(str)	Whether one contains other's characters at start
endsWith(str)	Whether one contains other's characters at end
contains(str)	Whether the given string is found within this one

[Table credit: Paerson Education]

Overriding equals() Methods

- Consider the following code:

```
class Complex {
    private double re, im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }
}

// Driver class to test the Complex class
public class Main {
    public static void main(String[] args) {
        Complex c1 = new Complex(10, 15);
        Complex c2 = new Complex(10, 15);
        if (c1 == c2) {
            System.out.println("Equal ");
        } else {
            System.out.println("Not Equal ");
        }
    }
}
```

[Code example from: GeeksForGeeks: Overriding equals method in Java, http://www.geeksforgeeks.org/overriding-equals-method-in-java/]

What is the output???

Overriding equals() Methods

- Consider the following code:

```
class Complex {

    private double re, im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }
```

[Code example from: GeeksForGeeks: Overriding equals method in Java, http://www.geeksforgeeks.org/overriding-equals-method-in-java/]

Overriding equals() Methods

- Consider the following code:

```
    // Overriding equals() to compare two Complex objects
    @Override
    public boolean equals(Object o) {

        // If the object is compared with itself then return true
        if (o == this) {
            return true;
        }

        /* Check if o is an instance of Complex or not
          "null instanceof [type]" also returns false */
        if (!(o instanceof Complex)) {
            return false;
        }

        // typecast o to Complex so that we can compare data members
        Complex c = (Complex) o;

        // Compare the data members and return accordingly
        return Double.compare(re, c.re) == 0
                && Double.compare(im, c.im) == 0;
    }
}
```

[Code example from: GeeksForGeeks: Overriding equals method in Java, http://www.geeksforgeeks.org/overriding-equals-method-in-java/]

Overriding equals() Methods

- Consider the following code:

```
// Driver class to test the Complex class
public class Main {

    public static void main(String[] args) {
        Complex c1 = new Complex(10, 15);
        Complex c2 = new Complex(10, 15);
        if (c1.equals(c2)) {
            System.out.println("Equal ");
        } else {
            System.out.println("Not Equal ");
        }
    }
}
```

[Code example from: GeeksForGeeks: Overriding equals method in Java, http://www.geeksforgeeks.org/overriding-equals-method-in-java/]

- What is the output now???

The Random Class

- A `Random` object generates pseudo-random* numbers
  - Class Random is found in the java.util package
- Example:

```
import java.util.*;
Random rand = new Random();
int randomNumber = rand.nextInt(25); // 0-24
```

Method	Description
nextInt()	Returns a random integer
nextInt(max)	Returns a random integer in the range [0, max]
nextDouble()	Returns a random real number in the range [0.0, 1.0]

[Table credit: Paerson Education]

Whirlwind Tour of Java

ASSERTIONS AND EXCEPTIONS

Logical Assertions

- Assertions – statements that are either true or false

- Examples:
  - Java is not the most popular programming language. → FALSE
  - 17 is a prime number. → TRUE
  - 25 is smaller than 625. → TRUE
  - X divided by 5 equals 4. → IT DEPENDS

Assertions in Code

- We can make assertions about our code and ask whether they are true at various points in the code
- Valid answers are ALWAYS, NEVER, or SOMETIMES

```
System.out.print("Type a nonnegative number: "); double number =
console.nextDouble();
// Point A: is number < 0.0 here?  (SOMETIMES)

while (number < 0.0) {
// Point B: is number < 0.0 here?   (ALWAYS)

System.out.print("Negative; try again: ");
number = console.nextDouble();
// Point C: is number < 0.0 here?   (SOMETIMES)

// Point D: is number < 0.0 here?  (NEVER)
```

Exceptions in Java

- Exception - a problem that arises during the execution of a program
  - *When an exception occurs, a normal flow of a program is disrupted, and a program terminates abnormally*

- Causes of exceptions:
  - User errors
  - Programmer errors
  - Failure of physical resources

Exception Categories in Java

- Exception - a problem that arises during the execution of a program
  - *When an exception occurs, a normal flow of a program is disrupted, and a program terminates abnormally*

- Three possible categories of exceptions:
  - **Checked exceptions** – an exception that occurs at the compile time (compile time exceptions)
  - Example: if you use **FileReader** class in your program to read data from a file, and if the file specified in its constructor doesn't exist, then a *FileNotFoundException* occurs, and the compiler prompts the programmer to handle the exception

Exception Categories in Java

- Exception - a problem that arises during the execution of a program
  - *When an exception occurs, a normal flow of a program is disrupted, and a program terminates abnormally*

- Three possible categories of exceptions:
  - Unchecked exceptions – an exception that occurs at the time of execution (**runtime exception)**
    - Include programming bugs, (e.g., logic errors or improper use of an API)
  - Example: if you have declared an array of size 5 in your program, and trying to call the 6th element of the array, an *ArrayIndexOutOfBoundsExceptionexception* occurs

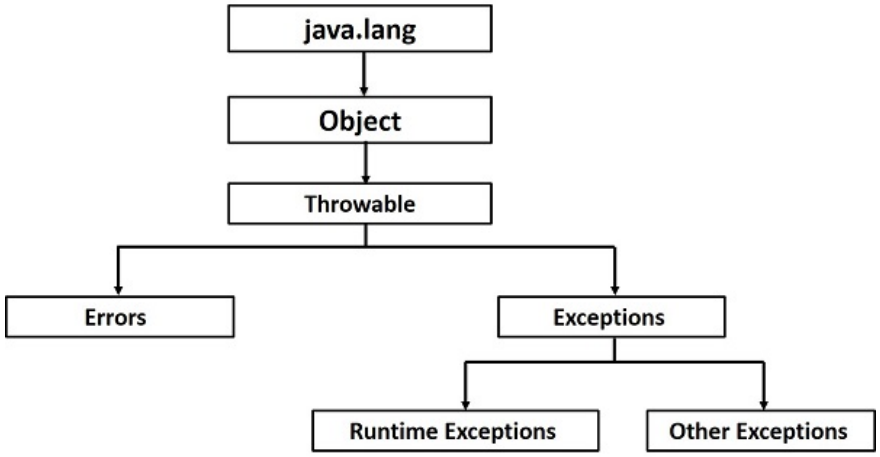
Exception Categories in Java

- Exception - a problem that arises during the execution of a program
  - *When an exception occurs, a normal flow of a program is disrupted, and a program terminates abnormally*

- Three possible categories of exceptions:
  - **Errors** – problems that arise beyond the control of the user or the programmer
  - Example: if a stack overflow occurs, an error will arise

Exceptions Hierarchy

- All exception classes are subtypes of the `java.lang.Exception` class
- The `Exception` class is a subclass of the `Throwable` class.
- The Exception class has two main subclasses:
  - `IOException` class
  - `RuntimeException` class



[Graphic credit: Java tutorial]

Catching Exceptions

- A method catches an exception using a combination of the **try** and **catch** keywords
  - *A try/catch block is placed around the code that might generate an exception (protected code)*

- Syntax:

```
try {
        // Protected code
}catch(ExceptionName e1)
    {
        // Catch block
    }
```

Catching Exceptions

- Syntax:

```
try {
        // Protected code
}catch(ExceptionName e1)
    {
        // Catch block
    }
```

- A catch statement declares the type of exception you are trying to catch
  - If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked
  - If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter

Exceptions Methods

- **public String getMessage()** - returns a detailed message about the exception that has occurred
- **public Throwable getCause()** - returns the cause of the exception as represented by a Throwable object
- **public String toString()** - returns the name of the class concatenated with the result of getMessage()
- **public void printStackTrace()** - prints the result of toString() along with the stack trace to System.err, the error output stream
- **public StackTraceElement [] getStackTrace()** - returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack
- **public Throwable fillInStackTrace()** - fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace

Whirlwind Tour of Java

TESTING WITH JAVA

Software Tests

- Software test - a piece of software, which executes another piece of software, in order to validate that code:
  - Results in the expected state (state testing)
  - Executes the expected sequence of events (behavior testing)

- Software unit tests – used to verify that the logic of a piece of the program is correct

Some Testing Terminology

- Unit test - a piece of code written by a developer that executes a specific functionality in the code to be tested, and asserts a certain behavior or state
  - *Not suitable for testing complex user interface or component interaction*
- Test fixture - fixed state in the tested code which is used as input for a test (test precondition)
- Test coverage - the percentage of code tested by unit test
- Integration test - a code used to test the behavior of a component, or the integration between a set of components
- Performance tests – test used to repeatedly benchmark software components

Testing Frameworks for Java

- Several testing frameworks available for Java:
- JUnit
- TestNG

Testing Frameworks for Java - Example

- The test assumes that the MyClass class exists and has a multiply(int, int) method.

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
public class MyTests {
    @Test
    public void
multiplicationOfZeroIntegersShouldReturnZero() {
            MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
            assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
    assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
    assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0)); } }
```

[Code example from: Vogela, Unit Testing with Junit]

Whirlwind Tour of Java

JAVADOC

Javadoc

- Javadoc – a tool for generating API documentation in HTML format from doc comments in source code

- Javadoc convention for writing specifications
  - Method signature
  - Text description of method:
    - @param: description of what gets passed in
    - @return: description of what gets returned
    - @throws: exceptions that may occur

Example: Javadoc for String.contains

public boolean contains(CharSequence s)
Returns true if and only if this string contains the specified sequence of char values.
Parameters:
    s- the sequence to search for
Returns:
    true if this string contains s,
    false otherwise
Throws:
    NullPointerException – if s is null
Since: 1.5

Javadoc & Pre/postconditions

- The *precondition*: constraints that hold before the method is called (if not, all bets are off)
  - @requires: spells out any obligations on client

- The *postcondition*: constraints that hold after the method is called (if the precondition held)
  - @modifies: lists objects that may be affected by method; any object not listed is guaranteed to be untouched
  - @throws: lists possible exceptions and conditions under which they are thrown (Javadoc uses this too)
  - @effects: gives guarantees on final state of modified objects
  - @return: describes return value (Javadoc uses this too)

Whirlwind Tour of Java

JAVA HASHCODE

Java hashCode()

- In Java, every class implicitly or explicitly provides a **`hashCode()`** method, which digests the data stored in an instance of the class into a single hash value (a 32-bit signed integer)
- Hash used by other code when storing or manipulating the instance
- This property is important to the performance of hash tables and other data structures that store objects in groups ("buckets") based on their computed hash values

Java hashCode()

```
public class Employee {
    int employeeId;
    String name;
    Department dept;

    // other methods would be in here

    @Override
    public int hashCode() {
        int hash = 1;
        hash = hash * 25 + employeeId;
        hash = hash * 12 + name.hashCode();
        hash = hash * 9 + (dept == null ? 0 : dept.hashCode());
}
```

[Code example from: Wikipedia, Java hashCode(), https://en.wikipedia.org/wiki/Java_hashCode()]

Whirlwind Tour of Java

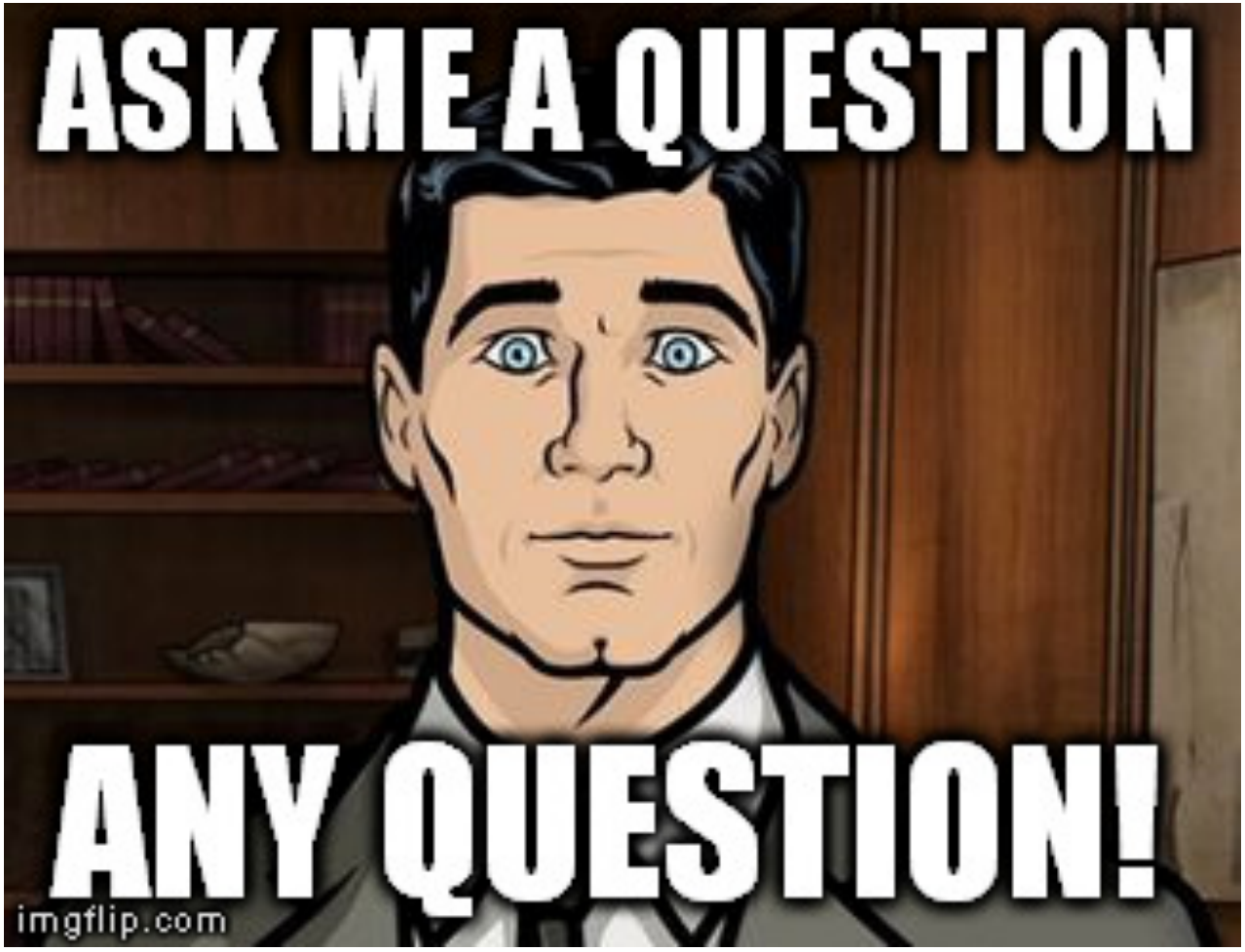
WHAT'S NEW IN JAVA 8 AND 9?*

* EXHAUSTIVE LIST AT HTTP://WWW.ORACLE.COM/TECHNETWORK/JAVA/JAVASE/8-WHATS-NEW-2157071.HTML AND HTTPS://DOCS.ORACLE.COM/JAVASE/9/WHATSNEW/TOC.HTM#JSNEW-GUID-71A09701-7412-4499-A88D-53FA8BFBD3D0

What's New in Java 8?

- Collections
  - Classes in the new **java.util.stream** package provide a Stream API to support functional-style operations on streams of elements
- Security
  - Many, many good things! Check them out ☺
- Javac tool
  - The parameters options of the **javac** command can be used to store formal parameter names
  - The **javac** tool now has support for checking the content of javadoc comments
- Javadoc tool
  - The **javadoc** tool supports the new DccTree API that enables you to traverse comments as abstract syntax trees

Your Questions



[Meme credit: imgflip.com]

References and Reading Material

- Java Tutorial: Java – Object and Classes, [Online] https://www.tutorialspoint.com/java/java_object_classes.htm
- Oracle Java Tutorial - Primitive Data Types, [Online], http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
- Java Tutorial: Java – Modifier Types, [Online], https://www.tutorialspoint.com/java/java_modifier_types.htm
- Java Programming Tutorial, Java String is Special, [Online],
- https://www.ntu.edu.sg/home/ehchua/programming/java/J3d_String.html
- Geeks For Geeks: Overriding equals Method in Java, [Online],
- http://www.geeksforgeeks.org/overriding-equals-method-in-java/
- Java Tutorial: Java – Exceptions, [Online], https://www.tutorialspoint.com/java/java_exceptions.htm
- Vogela, Unit Testing with Junit, [Online],
- http://www.vogella.com/tutorials/JUnit/article.html
- Oracle, What's New in JDK 8, [Online], http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html
- Oracle, Java Platform, What's New in JDK 9, [Online],
- https://docs.oracle.com/javase/9/whatsnew/toc.htm#JSNEW-GUID-71A09701-7412-4499-A88D-53FA8BFBD3D0