Objectives

- Collections wrap up
- Exceptions
- Eclipse

Review

- What are *wrapper* classes? When do we use them?
- What are the components of the Java Collections Framework?
- What are the three *interfaces* we discussed?
  - What are example *implementations* of those interfaces?
- I made the claim that this is the preferred way to create an object variable that adheres to an interface:

```
Interface variable = new Implementation();
Example: List<Card> hand = new ArrayList<>();
```

  - Why is that the preferred way? What is the design principle it adheres to?

ALGORITHMS

Collections Framework's Algorithms

- *Polymorphic algorithms*
- Reusable functionality
- Implemented in the `Collections` class
  - Similar to `Arrays` class, which operates on arrays
  - Static methods, 1st argument is the Collection

Overview of Available Algorithms

- Sorting – optional Comparator
- Shuffling
- Searching – binarySearch

* Only Lists

- Routine data manipulation: reverse*, copy*, fill*, swap*, addAll
- Composition – frequency, disjoint
- Finding min, max

TRAVERSING COLLECTIONS

Review: Traversing Collections: For-each Loop

- For-each loop:

Or whatever data type is appropriate

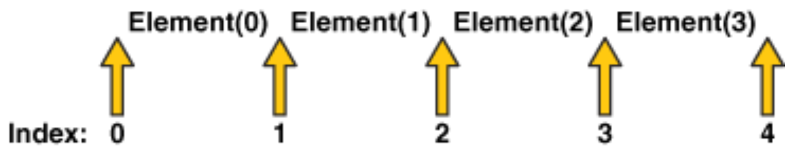
```
for (Object o : collection)
    System.out.println(o);
```

- Valid for all `Collections`
  - `Maps` (and its implementations) are not `Collections`
    - But, `Map`'s `keySet()` is a `Set` and `values()` is a `Collection`

7

Traversing Lists: Iterator

- Always between two elements



```
Iterator<Integer> i = list.iterator();
while( i.hasNext()) {
        int value = i.next();
        …
}
```

Helpful to use if removing elements from list during iteration

8

Benefits of Collections Framework

- ?

Benefits of Collections Framework

- **Provides common, well-known interface**
  - Allows interoperability among unrelated APIs
  - Reduces effort to learn and to use new APIs for different implementations
- **Reduces programming effort:** provides useful, reusable data structures and algorithms
- **Increases program speed and quality:** provides high-performance, high-quality implementations of data structures and algorithms; interchangeable implementations → tuning
- **Reduces effort to design new APIs:** use standard collection interface for your collection
- **Fosters software reuse:** New data structures/algorithms that conform to the standard collection interfaces are reusable

EXCEPTIONS

Error Handling

- Programs encounter errors when they run
  - Users may enter data in the wrong form
  - File may not exist
  - Program code has bugs!*
- When an error occurs, a program should do one of two things:
  - Revert to a stable state and continue
  - Allow the user to save data and then exit the program gracefully

* (Of course, not *your* programs)

Java Method Behavior

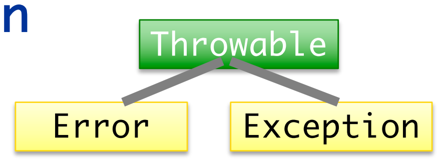
- Normal/correct case: return specified return type
- Error case: does not return anything, `throws` an `Exception`
    - An ***exception*** is an event that occurs during execution of a program that disrupts normal flow of program's instructions
    - `Exception`: object that encapsulates error information

Similar to Python

19

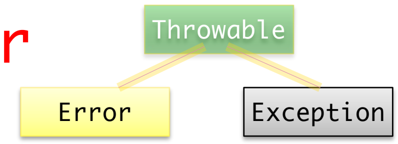
Throwable

- All exceptions indirectly derive from **`Throwable`**
    - Child classes: **`Error`** and **`Exception`**
- Important **`Throwable`** methods
    - `getMessage()`
        - Detailed message about error
    - `printStackTrace()`
        - Prints out where problem occurred and path to reach that point
    - `getStackTrace()`
        - Get the stack in non-text format



20

Exception Classification: Error



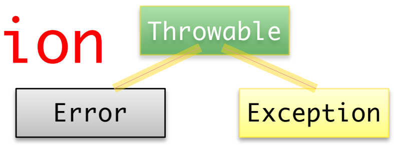
- An internal error
- Strong convention: reserved for JVM
  - JVM-generated when resource exhaustion or an internal problem
    - Example: Out of Memory error

When can that happen in Java?

- Program's code should not and can not throw an object of this type
- This is an example of an *Unchecked* exception

21

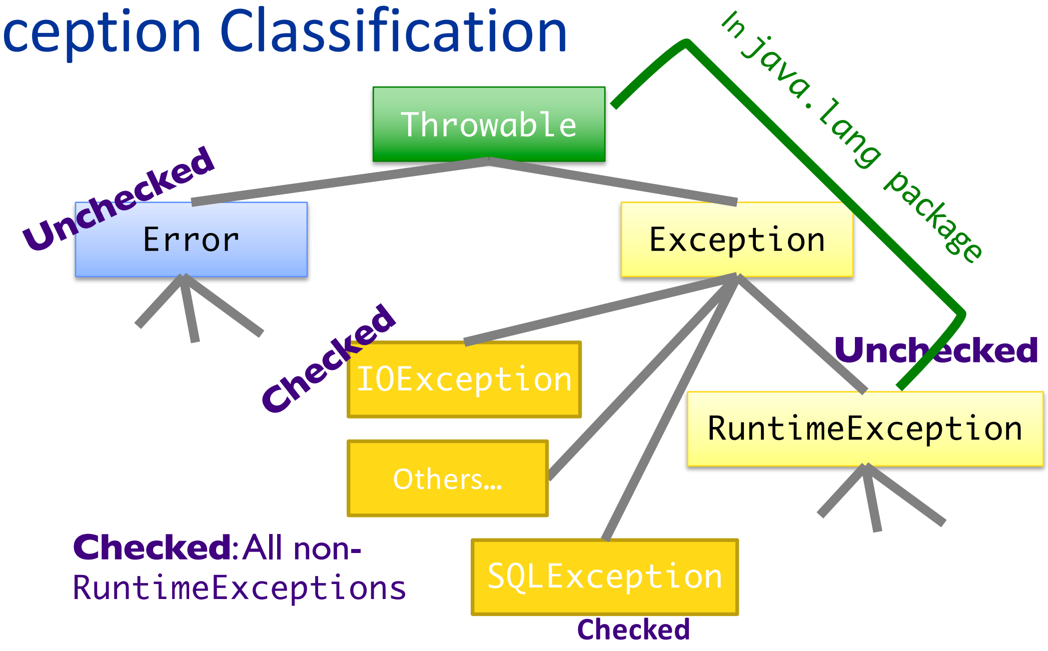
Exception Classification: Exception



1. `RuntimeException`:
   something that happens because of a programming error
   - **Unchecked** exception
   - Examples: `ArrayOutOfBoundsException, NullPointerException, ClassCastException`
2. **Checked** exceptions
   - A well-written application should anticipate and *recover* from these exceptions
   - Compiler enforces that programmer handles them
   - Examples: **`IOException, SQLException`**

22

Exception Classification



Checked: All non-`RuntimeExceptions`

Categories of Exceptions

Unchecked

- Any exception that derives from `Error` or `RuntimeException`
- Programmer does not necessarily create/handle
- **Try to prevent RuntimeExceptions**
  - Often indicates programming error
  - E.g., precondition violations, not using API correctly, dividing by 0

Checked

- Any other exception
- For conditions from which caller can reasonably be expected to recover
- Compiler-enforced checking
  - Program MUST handle
  - Improves *reliability**

Types of Unchecked Exceptions

1. Derived from the class `Error`
   - Any line of code can generate because it is an internal JVM error
   - Don't worry about what to do if this happens
2. Derived from the class `RuntimeException`
   - Indicates a bug in the program
   - Fix the bug, try to prevent
   - Examples: `ArrayOutOfBoundsException`, `NullPointerException`, `ClassCastException`

Checked Exceptions

- Need to be handled by your program
  - Compiler-enforced
  - Improves reliability*
- For each method, tell the compiler:
  - What the method returns
  - What could possibly go wrong
    - *Advertise* the exceptions that a method throws
    - Helps users of your interface know what method does and lets them decide how to handle exceptions

THROWING EXCEPTIONS

Methods and Exceptions Example

- `BufferedReader` has method `readLine()`
  - Reads a line from a *stream*, such as a file or network connection
- Method header:

$$\overbrace{\texttt{throws IOException}}^{\text{Part of Advertising}}$$

```
public String readLine() throws IOException
```

- Interpreting the header: `readLine` will
  - return a String (if everything went right)
  - throw an `IOException` (if something went wrong)

Advertising Checked Exceptions

- Advertising in Javadoc: document under what conditions each exception is thrown
  - `@throws` tag
- Examples of when your method should advertise the ***checked*** exceptions that it may throw
  - Your method calls a method that throws a checked exception
  - Your method detects an error in its processing and decides to throw an exception

29

Example: Passing an Exception "Up"

```
public String readData(BufferedReader in)
    throws IOException {
        String str1 = in.readLine();
        return str1;
}
```

Throws an `IOException`

- `readData` calls `readLine`, which can throw an `IOException`
- If `readLine` throws this exception to our method
  - `readData` *throws* the exception as well
  - Whoever calls `readData` will handle exception

30

Example: Throwing An Exception We Created

1. Create a new object of class **IllegalArgumentException**
   - Class derived from **RuntimeException**
2. throw it
   - Method ends at this point
   - Calling method handles exception

```
if (grade < 0 || grade > 100) {
        throw new IllegalArgumentException();
}
```

Equivalent in Python?

A More Descriptive Exception

- Four constructors for most Exception classes
  - Default (no parameters)
  - Takes a String message
    - Describe the condition that generated this exception more fully
  - 2 more

```
if (grade < 0 || grade > 100) {
        throw new IllegalArgumentException(
                "Grade is not in valid range (0-100)");
}
```

Best messages include all state that could have contributed to the problem

Common Exception Classes

Name	Purpose
`IllegalArgumentException`	When caller passes in inappropriate argument
`IllegalStateException`	Invocation is illegal because of receiving object's state. (Ex: closing a closed window)

- Both inherit from `RuntimeException`
- May seem like these cover everything but only used for certain kinds of illegal arguments and exceptions
- Not used when
  - A null argument passed in; should be a `NullPointerException`
  - Pass in invalid index for an array; should be an `IndexOutOfBoundsException`

33

Birthday Error Handling Discussion

- Design decision:
  - Since month and day are not independent, should be set *together* rather than separately
- Check all the error cases before setting the instance variables
  - Don't want an inconsistent resulting birthday
- `IllegalArgumentException` is appropriate
  - Programming error
  - Should catch those errors before executing program

34

Goal: Failure Atomicity

- After an object throws an exception, the object should be in a well-defined, usable state
  - A failed method invocation should leave object in state prior to invocation
- Approaches:
  - Check parameters/state before performing operation(s)
  - Do the failure-prone operations first
  - Use recovery code to "rollback" state
  - Apply to temporary object first, then copy over values

Javadoc Guidelines about @throws

- Always report if throw ***checked*** exceptions
- Report any unchecked exceptions that the caller might reasonably want to catch
  - Exception: `NullPointerException`
  - Allows caller to handle (or not)
  - Document exceptions that are independent of the underlying implementation
- `Errors` should **not** be documented as they are unpredictable

eclipse

eclipse https://www.eclipse.org/

- Open source integrated development environment (IDE) for Java
- Described as "an open extensible IDE for anything and nothing in particular"
- Provides a robust Java development environment
- Incorporates popular software development tools like JUnit and git
- Plugins allow extensibility

Project/Code Organization

- `workspace` directory contains all projects
  - Located in your home directory, unless you specified otherwise
- Use ***projects*** to organize your code
- Within a project
  - `src/` directory contains `.java` files
  - `bin/` directory contains `.class` files
    - Often hidden in GUI

Oct 19, 2022 Sprenkle - CSCI209 39

39

Java Made Easier

- Creating class's basic functionality
  - See Source and Refactor menus
- Gives you a list of methods for an object
  - After you type object.
  - Then shows parameters for methods
- Automatically creates template of Javadoc
  - When you type /**
- Autocompletion of variables, methods
- Formatting code …
- Shows unused fields/variables
- Shows compiler errors
- …

Oct 19, 2022 Sprenkle - CSCI209 40

40

Eclipse Demo

- Create a new `Birthday` class
  - Generate `main` method, Comments
- Demonstrate Source menu
  - Generate constructor, toString
  - Override `equals` method
- Create a String object, see methods used
- Demonstrate Refactor menu
  - Rename a field
  - Extract a method (month name)
- Run the Birthday Class (main)
  - Command line arguments
- Using git

Why can a Java IDE provide this functionality?

Eclipse Hints

- After you have written a method, type

`/**`

before the method, and then hit enter and the Javadocs comment template will be automatically generated for you

- Use `command-spacebar` for possible completions
- Use `command-shift-F` to format code

Eclipse Tradeoffs

- Very helpful – *after* you know what you're doing
  - You know
    - Code is compiled before executed
    - Structure of classes
    - How to fix errors
- Eclipse can handle the "routine" for you
  - That wasn't "routine" for you a few weeks ago
  - Help you focus on the important design considerations
- Gives suggestions for fixes
  - **You need to think through what the appropriate fix is**
    - Sometimes, it's "I'm not done yet"
  - Don't say "Eclipse made me do <something>"
- Eclipse is a beast (memory hog)
  - If you have less than ~8GB of memory, Eclipse will be slow

Looking Ahead

- Eclipse set up for Friday
- Change in Thursday office hours: 10:30 a.m. - 12:30 p.m.
  - Updated in Canvas site on Calendar