6.092 - Introduction to Software Engineering in Java

Lecture 8:

Exceptions, I/O, and you++

Thursday, January 31
IAP 2008

Administrivia

HKN Course Evaluations: Site is now live!

- Help us improve, help students choose
- Survey website is now active
- Only active for a few days, so do it soon
- https://sixweb.mit.edu/

Course Refresher

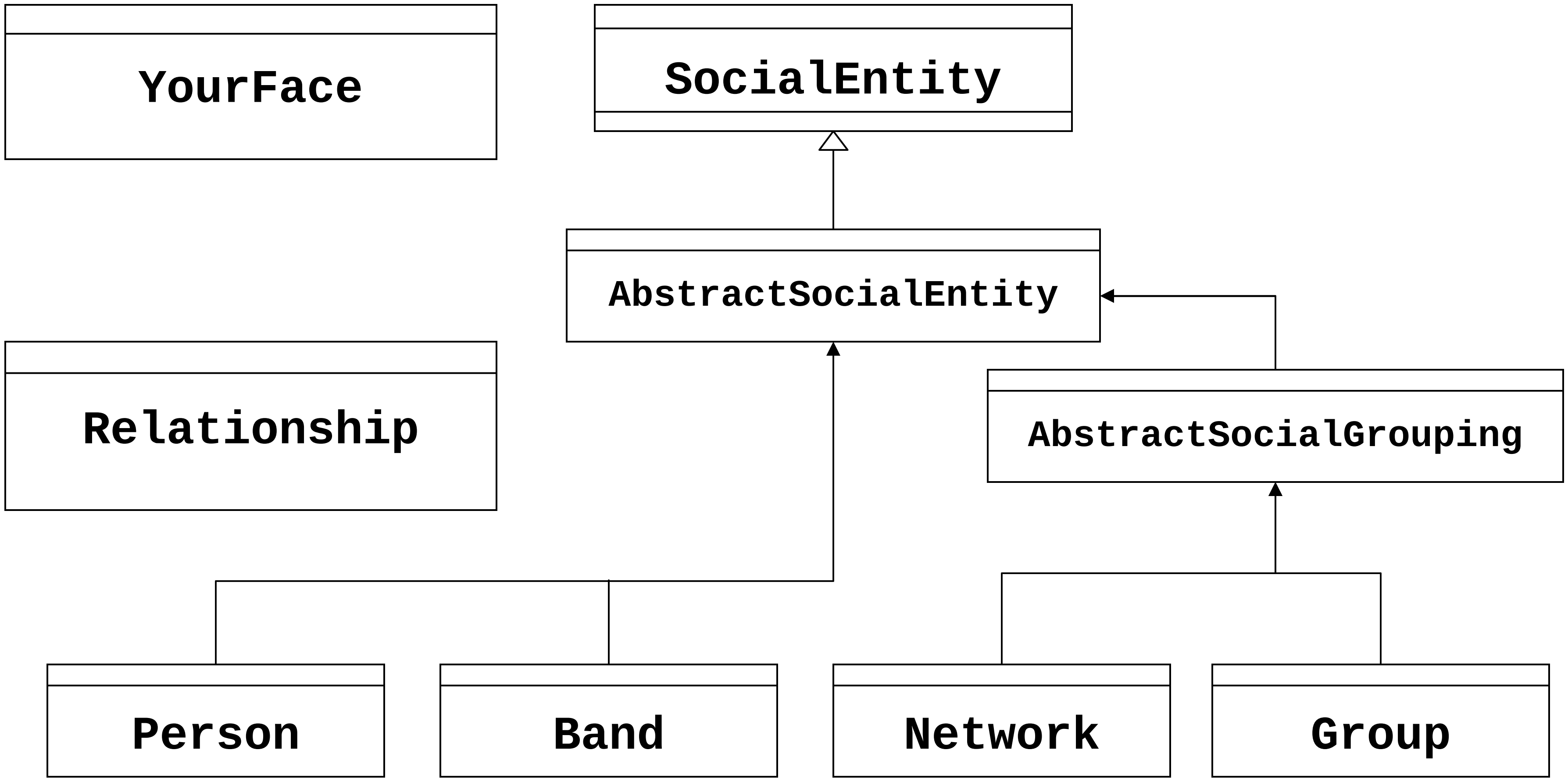
- What you've learned so far
  - Intro/Overview
    - compilation, execution
  - Java Basics:
    - Structure & Syntax, Variables, Types, & Operators
  - Control Flow:
    - Methods & Conditionals, Loops & Arrays
  - Object-oriented Programming (OOP):
    - Objects, Classes, Interfaces, Inheritance, Abstraction, Encapsulation
  - Brief Intro to Software Design
  - More Useful Tools
    - Packages, Collections, the Java API

Review: Assignment 7

Refining YourFace: A Simple Social Network

- Adding abstract classes
- Using them to reduce duplicate code
- Converting arrays to Collections
- Using packages to logically group related classes

Assignment 7: Example Diagram



Ideal Solution: `AbstractSocialEntity`

```java
package yourface2.entities;

public abstract class AbstractSocialEntity implements SocialEntity {

  protected final String name;
  protected final long id;

  protected AbstractSocialEntity(String name, long id) {
    this.name = name;
    this.id = id;
  }

  public String getName() {
    return this.name;
  }

  public long getId() {
    return this.id;
  }

}
```

Ideal Solution: `AbstractSocialGrouping`

```
package yourface2.entities;

import java.util.HashSet;
import java.util.Set;

public abstract class AbstractSocialGrouping extends AbstractSocialEntity {

  protected final Set<Person> members;

  protected AbstractSocialGrouping(String name, long id) {
    super(name, id);
    this.members = new HashSet<Person>();
  }

  public Set<Person> getMembers() {
    return this.members;
  }

  public void addMember(Person p) {
    this.members.add(p);
  }

}
```

Ideal Solution: `Relationship`

```
package yourface2.entities;

public class Relationship {

  private final String type;
  private final String decsription;
  private final boolean isMutual;

  public static final Relationship
    ACQUAINTANCE = new Relationship("acquaintance", true),
    FRIEND = new Relationship("friend", true),
    COWORKER = new Relationship("coworker", true),
    RELATIVE = new Relationship("relative", true),
    STUDENT = new Relationship("student", false),
    TEACHER = new Relationship("teacher", false);

  public Relationship(String type, boolean isMutual, String description) {
    this.type = type;
    this.isMutual = isMutual;
    this.decsription = description;
  }

  public Relationship(String type, boolean isMutual) {
    this(type, isMutual, "is a "+ type + " of");
  }

  // ... getters

  public String toString() {
    return getDescription();
  }
```

Ideal Solution: Person

```
package yourface2.entities;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class Person extends AbstractSocialEntity {

  protected Map<Person,Relationship> acquaintances;
  protected Set<Network> networks;
  protected String location;

  public Person(String name, long id, String location) {
    super(name, id);
    this.location = location;
    this.acquaintances = new HashMap<Person, Relationship>();
    this.networks = new HashSet<Network>();
  }

   // more ...

}
```

Ideal Solution: Person

```
package yourface2.entities;
// ... imports
public class Person extends AbstractSocialEntity {

  // overloading, adding auto-adding mutual relationships
  public void addAcquaintance(Person p, Relationship r) {
    this.acquaintances.put(p, r);
    if (r.isMutual()) {
      p.getAcquaintances().put(this, r);
    }
  }

  public void addAcquaintance(Person p) {
    this.addAcquaintance(p, Relationship.ACQUAINTANCE);
  }

  public Relationship getRelationship(Person p) {
    return this.acquaintances.get(p);
  }

}
```

Ideal Solution: Person

```
package yourface2.entities;

// ... imports

public class Person extends AbstractSocialEntity {
  // ... more

  public Set<Network> getNetworks() {
    return this.networks;
  }

  public void addNetwork(Network network) {
    this.networks.add(network);
  }

  public String toString() {
    String acqsToString = "\n Acquaintances: ";
    for (Person p : acquaintances.keySet()) {
      acqsToString +=
        "\n  "+p.getName()+" "+getRelationship(p)+" "+this.getName();
    }
    String netsToString = "\n Networks: ";
    for (Network n : networks) {
      netsToString += "\n  "+n.getName();
    }
    return "Person #"+getId()+": "+getName()+
           "\n Location: "+getLocation()+
           acqsToString+netsToString;
  }

}
```

Ideal Solution: YourFace2

```
package yourface2;
import yourface2.entities.Network;
import yourface2.entities.Person;
import yourface2.entities.Relationship;
import yourface2.entities.SocialEntity;

public class YourFace2 {

  public static void main(String[] args) {
    // ... Creating Persons omitted here
    // add Acquaintances
    usman.addAcquaintance(evan, Relationship.COWORKER);
    usman.addAcquaintance(olivier, Relationship.COWORKER);
    usman.addAcquaintance(student1, Relationship.STUDENT);
    evan.addAcquaintance(olivier, Relationship.COWORKER);
    evan.addAcquaintance(student2, Relationship.STUDENT);
    olivier.addAcquaintance(student1, Relationship.STUDENT);
    student1.addAcquaintance(usman, Relationship.TEACHER);
    student1.addAcquaintance(olivier, Relationship.TEACHER);
    student1.addAcquaintance(student2, Relationship.FRIEND);
    student2.addAcquaintance(evan, Relationship.TEACHER);

    // ... then easy stuff
    printArray(new Person[]{usman, evan, olivier, student1, student2});

  }
}
```

Assignment 7: Recap

- Reminders:
  - abstract classes can and should have constructors & fields
  - Map is not iterable
    - use keySet() or entrySet or values()
- Caveats:
  - when printing a Person's acquaintances, avoid infinite loops!

Today’s Topics

- Exceptions
- Input/Output
- You++ : What’s next?

Exceptions

- A way to tell when something goes wrong in a method call
- When an error happens, an Exception object is ***thrown***
- You’ve already seen them
- Useful for debugging & control flow

Exceptions: Types

- Common types of Exceptions
  - RuntimeExceptions
    - NullPointerException
    - ClassCastException
    - ArrayIndexOutOfBoundsException
    - Etc.
  - Other Exceptions

Exceptions: Basic Usage

- To declare that you throw an exception:

```
public Object pop() throws EmptyStackException {
    Object obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

Exceptions: Basic Usage

- Using a method that throws an Exception
  - ***try*** it
  - If it doesn't work, it will throw its Exception
  - Then you must ***catch*** the exception
  - You can catch multiple Exception types
  - Example:

```
try {
     // do something with a File
} catch (FileNotFoundException e) {
    System.err.println("FileNotFoundException: “
                      + e.getMessage());
    throw new SampleException(e);

} catch (IOException e) {
    System.err.println("Caught IOException: “
                       + e.getMessage());
}
```

Java I/O

- Data can flow in streams

- You can Read from (input) or Write (output) to a stream

Java I/O

- Input
  - System.in
  - Network
  - File
  - Etc.

- Output
  - System.out
  - System.err
  - Network
  - File
  - Etc.

Java I/O

- Ways to access data
  - Streams
  - Readers
  - Writers

- These can also be ***Buffered***

Java I/O: A Tour

http://java.sun.com/javase/6/docs/api/index.html?java/io/package-summary.html

Java I/O: Example

Reading Text from a file:

```
try {
  BufferedReader in =
    new BufferedReader(new FileReader("infilename"));
  String str;
  while ((str = in.readLine()) != null) {
    process(str);
  }
  in.close();
} catch (IOException e) {
  // handle the potential exception
}
```

You++ : What’s next?

- Course 6
- The “Real World”

You++ : Course 6

- Software Engineering (e.g. 6.005)
  - design patterns
  - teamwork
  - discipline
  - using other libraries for development
  - GUIs (or not)

- Computer Science (e.g. 6.042, 6.046)
  - Algorithms
  - Math & Proofs
  - notation

You++ : The “Real World”

- “Software Development Lifecycle”
  - Design
    - SW eng. Principles, design patterns
    - tools: whiteboards, powerpoint, sipb:xfig
  - Document
    - EXACT specs lead to a better PROGRAM, even if code stinks
    - tools: javadoc, design docs, etc.
  - Build
    - implementation - languages du jour are Java & python; know them
    - tools: emacs, eclipse, etc.
  - Test
    - different from debugging; tools: JUnit, LOTS of others
- science & engineering

Assignment 8: *Hai*

- ***Hai***: a simple chat server & client
  - Use the Java API, finish the code

Assignment 8: *Hai*

- Learn to use Exceptions
- See the basics of I/O in action
- Learn to learn from documentation

Assignment 8: *Details*

- Fill in the TODOs!
- Catch and handle exceptions.
- Get appropriate input & output streams for the SERVER.
- Get appropriate input & output streams for the CLIENT.
- Create the appropriate Reader & Writer from each stream.

Assignment 8: *Hai*

- ***Tips:***
  - Start ***now***
  - Stay for the lab hour
  - Ask questions often (in person or via email)
  - Reuse your old code as much as you can
  - Use the Java API!
    - Along with other references listed on the course homepage
  - ***This one is really due on Friday at 4***

`return you++; // :)`

- Hope you enjoyed 6.092!
  - Now enjoy tasty treats!