Announcements

- Exam 1 Monday, February 28
  - Wetherill 200, 4:30pm-5:20pm
  - Coverage: Through Week 6
    - Project 2 is a good study mechanism
- Final Exam
  - Tuesday, May 3, 3:20pm-5:20pm, PHYS 112
  - If you have three or more finals that day, you may contact me about rescheduling

PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

Control Flow: Exceptions

21 February 2011

Prof. Chris Clifton

COMPUTER SCIENCES

What do you do when something goes wrong?

- Problem: Things don't always go as expected
  - Invalid arguments to a method
  - Can't do what is requested *in this state*
  - …
- Sometimes we can provide an error message
  - But this doesn't always make sense

Example: Queue

```
interface Queue {
    void enqueue (Person item);
    Person dequeue();
    boolean isEmpty();
}

…

s.dequeue().learn
  ("Exceptions");
```

```
class StupidQueue
  implements Queue {
    private int head, tail;
    private Person q[];
    public void enqueue( Person
      item ) {
        q[tail++] = item;
        }
    public Person dequeue() {
        return q[head++];
        }
    public boolean isEmpty() {
    … }
}
```

Problem: What if nobody in the queue?

```
interface Queue {
    Person EMPTYRESULT =
      null;
    void enqueue (Person item);
    Person dequeue();
    boolean isEmpty();
}

…

s.dequeue().learn
  ("Exceptions");
```

```
class StupidQueue
  implements Queue {
    private int head, tail;
    private Person q[];
    public void enqueue( Person
      item ) {
        q[tail] = item;
        q[++tail] = EMPTYRESULT  }
    public Person dequeue() {
        return q[head++];
        }
    …
}
```


Solutions

0. ~~Just it crash~~

1. Preconditions forbid misuse
   - May be okay for programmers
   - But what about user input?
2. Error return code
   - Check if empty
   - If it is, return special value

```
interface Queue {
    Person EMPTYRESULT =
      null;

    void enqueue (Person
      item);

    Person dequeue();
    /* Precondition:
      !isEmpty(); */

    boolean isEmpty;
}
```

Solutions

2. Error return code
   - Check if empty
   - If it is, return special value
     - Must check for special value when called
   - *Only works if the return type has "room" for error codes*
     - bq = new BooleanQueue;
3. Exceptions
   - End function without "return"ing

```
interface Queue {
    Person EMPTYQUEUE =
      null;

    void enqueue(Person item);

    Person dequeue();
            throws EmptyQueue;
    /* Precondition:  !isEmpty(); */

    boolean isEmpty;
}
```


Handling an Exception

- Method can't return normally
  - You won't be able to used a returned result
- Instead, continue someplace else
- try { … } catch
- So what is an EmptyQueue?
  - A type (Class) that extends Exception

```
Queue officehour;
…
try {
    officehour.dequeue().
      learn("Exceptions");
} catch (EmptyQueue
  e) {
    professor.getCoffee
      ("The Port");
}
```

Declaring an Exception

- To use an exception you must pick one
  - or define your own
- Can extend Exception without doing anything
  - Gives a unique name for your exception
- Or can add information
  - Reason it was generated
  - …
- If specific to a class, make it a nested class
  - Queue.EmptyQueue

```
interface Queue {
    public class EmptyQueue
       extends Exception { };

    void enqueue (Person item);

    Person dequeue() throws
       EmptyQueue;
    /* Precondition:  !isEmpty(); */

    boolean isEmpty;
}
```


Generating an Exception

- First, create an instance of the exception
  - May contain error message
  - Sample of the bad input
  - …
- Then "throw" it

```
public Person
   dequeue()
   throws EmptyQueue {
    if (isEmpty()) throw
       new EmptyQueue();
    // Else queue isn't empty
    return q[head++];
}
```

When to use exceptions *(guidelines)*

Do use

- In unexpected situations
- No meaningful return value
- Can't recover/process inside a method
- Doesn't make sense to continue after "failed" method call

Don't use

- Under normal conditions
- Possible to return something sensible
- Possible to recover and complete method
- The next steps are the same regardless of the exception