

A Necessary Digression to Necessary Java



Topics

- ◆ Interfaces
- ◆ Exceptions
- ◆ Casting

Interfaces

- ◆ An interface declares functionality but not the details of implementation
- ◆ Any class that implements an interface must provide those details
- ◆ The separation of interface from implementation is a good example of object-oriented design

Interface vs. Implementation

- ◆ An interface contains method declarations with
 - method name
 - return type
 - parameters and their types
- ◆ The implementing class contains code for the methods specified in the interface
- ◆ If a class implements an interface without defining these methods, then it must be declared abstract

```
public interface Radio {  
    public void play();  
    public void stop();  
}
```

```
public class BrownRadio  
    implements Radio {  
    public void play() {  
        System.out.println("You are listening  
            to WBRU this is  
            DJ Sasha!!!");  
    }  
    public void stop() {  
        System.out.println("Whew!");  
    }  
}
```

Exceptions

- ◆ An exception is a signal to indicate the occurrence of an *exceptional* condition (e.g., an error)
- ◆ To throw an exception is to indicate such an occurrence; when we find an error, we throw an exception

```
public class TA {  
    ...  
    public void eatPizza  
        throws StomachAcheException {  
        ...  
        if(eatTooMuch) {  
            throw new StomachAcheException("Bleh!");  
        }  
    }  
}
```

Note: if a method throws an exception, then a throws clause must be added to the method declaration

Exceptions (2)

- ◆ When an exception is thrown, the flow of control exits from the current scope
- ◆ So if `_stupidTA.eatPizza()` throws a `StomachAcheException`, we exit from the method `eatPizza()` and return to the line of code that invoked it in `simulateMeeting()`

Code fragment invoking `eatPizza()`

```
private TA _stupidTA;
...
public void simulateMeeting() {
    try {
        _stupidTA.eatPizza();
    }
    catch(StomachAcheException e) {
        System.out.println("Stupid TA has a
            stomach ache!");
    }
}
```

Exceptions: try & catch

- ◆ a **try** clause denotes a block of code that handles exceptions
- ◆ a **try** block can be followed by a **catch** clause that handles a specific type of exception

```
try {  
    _stupidTA.eatPizza();  
}  
catch(StomachAcheException e) {  
    System.out.println("Stupid TA has a  
                        stomach ache!");  
}
```

Note that **catch** is listening for `StomachAcheException`, so the flow of control enters the **catch** block and `System.out.println` is executed.

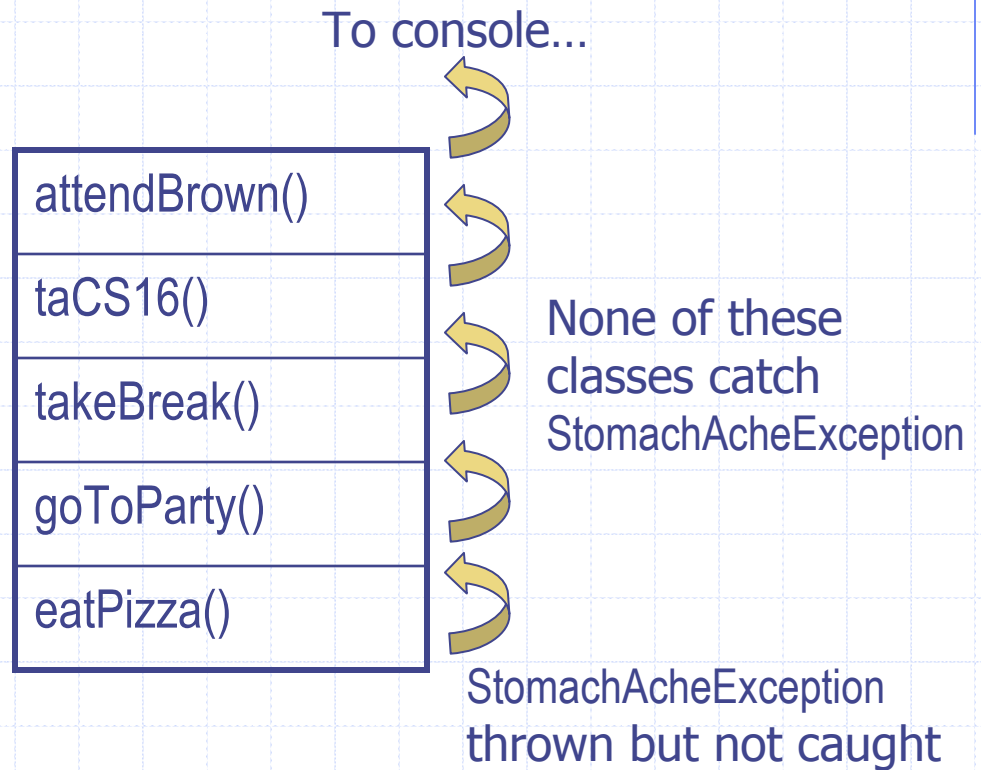
Exceptions: try & catch (2)

- ◆ Note that the **catch** block can contain any code to handle the exception
- ◆ E.g., we can handle the exception caught by throwing another exception

```
try {  
    _stupidTA.eatPizza();  
}  
catch(StomachAcheException e) {  
    throw new AlertMommyException("Help!");  
}
```

Exceptions: try & catch (3)

- ◆ Note also that no **catch** block must be specified
- ◆ If an exception is not caught, it propagates up through
 - the blocks of the current method
 - the chain of invoking methodsuntil the user sees it



What are Exceptions?

- ◆ Ok, so we try, throw, and catch exceptions
 - But what are they in Java?
 - Classes, of course

```
public class StomachAcheException extends RuntimeException {  
    public StomachAcheException(String err) {  
        super(err);  
    }  
    ...  
}
```

Generics

- ◆ In CS16, we write data structures to be generic, so that they can hold any sort of data.
- ◆ But we still want to be able to use the data as if it were the specific type that it actually is.
- ◆ Remember to Use Generics!
(PizzaDex, anyone?)

Casting and Exceptions

- ◆ What if we want to determine if a certain piece of data is valid (is an instance of a certain type)?
 - We can use **try & catch!**
 - We **try** to cast the data and **catch** an exception if one is thrown.
 - If an invalid cast is performed, Java will throw a `ClassCastException`.

Example: ClassCastException

```
public void AnalyzeData(GenericData gdata) {  
    SpecificData sdata = null;  
    try {  
        sdata = (SpecificData)gdata;  
    }  
    catch(ClassCastException e) {  
        throw new InvalidDataException("Data is not an instance of SpecificData");  
    }  
}
```

More Casting

- ◆ We can cast to a specific type and call a method on that type in one line of code!

```
public void AnalyzeData(GenericData gdata) {  
    ((SpecificData)gdata).specificMethod();  
}
```

- Note that the following code (which lacks parentheses) will not compile because `specificMethod()` is not a member of the `GenericData` class.

```
public void AnalyzeData(GenericData gdata) {  
    (SpecificData)gdata.specificMethod();  
}
```