Lecture 14
Exception Handling

• What is an exception?
  – Unexpected (or at least unusual or abnormal)
  – Disruptive
  – Possibly fatal

• Exception handling
  – Formal mechanism for detecting and dealing with exceptions
    • Most modern OO languages
      – Java, C++, C#, Python, Ruby, etc.
Before exception handling...

• What did we do before exception handling?
  – Error code
    • Specific return value of a function or procedure would be interpreted as indicating an error.
    • Some old conventions:
      – Procedures (i.e., would normally return nothing)
        » 0 = success, < 0 = error code
      – Functions (i.e., returns some sort of value)
        » If returning an object, null = error, otherwise choose a value “out of domain” of function to indicate error.
      – Global error code to distinguish specific type of error.
Revisiting Belt

• `getSushiAt()`, `takeOffSushiAt()`
  – Return null if position is actually empty.

• `placeSushiNearestTo()`
  – Returns -1 if belt is full
Exception Handling in Java

- Three kinds of exception:
  - Checked exceptions
    • Generally valid situations that application should be able to anticipate and deal with.
  - Error
    • External conditions that application generally can’t anticipate of do anything about.
  - Runtime exceptions
    • Internal conditions that usually indicate an error in logic
Raising Exceptions

• When exception occurs:
  – Create an object from Throwable hierarchy to represent the exception.
    • Provides information about the exception.
  – Use “throw” to raise the exception.
    • Throw and raise generally synonymous when talking about exception handling.
Specifying Checked Exceptions

• Checked exceptions must be specified as part of method signature.
  – Checked exceptions are any exceptions that are a subclass of Exception but not Error or RuntimeException.
  – Must indicate specific type of exception object that will be thrown.
  – After parameter list, before method body.
    • Keyword “throws” followed by comma separated list of possible exception object types.

• Example:
  ```java
  public Sushi getSushiAt(int pos)
      throws BeltPositionEmptyException {
      ...
  }
  ```
Catch or Specify Requirement

• Any method that calls another method that could throw a checked exception must either:
  – “catch” the exception, or...
  – “specify” the exception
  • Declare that this method could result in the exception.
    – Even thought this method didn’t throw it.
  • Compile-time requirement for Java
    – Won’t even run.
Catching Exceptions

• try – catch – finally

```java
try {
    // Code here cause an exception.
}
catch (ExceptionTypeA name) {
    // Code here executes if ExceptionTypeA is raised
    // Variable “name” is set to exception object.
}
catch (ExceptionTypeB name) {
    // Code here executes if ExceptionTypeB is raised
    // Variable “name” is set to exception object.
}
finally {
    // Code here always gets executed.
}
```
Putting It Together

• lec14.v1
• lec14.v2
Runtime Exceptions

• Runtime exceptions usually indicate a problem in logic.
  – Should avoid them rather than trying to handle them after the fact.
  • Where in our example could we do this?
General Principle: Be Specific

• Use an existing exception type.
  – There are lots.
  – If semantics of the exception match well, then go ahead and use it.

• Create your own exception type.
  – Subclass either RuntimeException or Exception
General Principle: Catch Late

• Exceptions should rise to level where application has enough context to deal with them effectively.