Today: Various odds and ends in support of abstraction.

Readings: At this point, we have looked at Chapters 1–9 of *Head First Java*. Today’s lecture is about Chapters 9 and 11. For Friday, please read Chapter 10 of *HFJ* and Chapter 3 (“Numbers”) from *Assorted Material on Java*. 
Parent constructors

• In lecture notes #5, talked about how Java allows implementer of a class to control all manipulation of objects of that class.

• In particular, this means that Java gives the constructor of a class the first shot at each new object.

• When one class extends another, there are two constructors—one for the parent type and one for the new (child) type.

• In this case, Java guarantees that one of the parent’s constructors is called first. In effect, there is a call to a parent constructor at the beginning of every one of the child’s constructors.

• You can call the parent’s constructor yourself. By default, Java calls the “default” (parameterless) constructor.

```java
class Figure {
    public Figure (int sides) {
        ...
    }...
}
class Rectangle extends Figure {
    public Rectangle () {
        super (4);
    }...
}
```
What to do About Errors?

- Large amount of any production program devoted to detecting and responding to errors.
- Some errors are external (bad input, network failures); others are internal errors in programs.
- When method has stated precondition, it's the client's job to comply.
- Still, it's nice to detect and report client's errors.
- In Java, we throw exception objects, typically:
  ```java
  throw new SomeException (optional description);
  ```
- Exceptions are objects. By convention, they are given two constructors: one with no arguments, and one with a descriptive string argument (which the exception stores).
- Java system throws some exceptions implicitly, as when you dereference a null pointer, or exceed an array bound.
Catching Exceptions

- A **throw** causes each active method call to **terminate abruptly**, until (and unless) we come to a **try** block.

- Catch exceptions and do something corrective with **try**:

```java
try {
    Stuff that might throw exception;
} catch (SomeException e) {
    Do something reasonable;
} catch (SomeOtherException e) {
    Do something else reasonable;
}
Go on with life;
```

- When **SomeException** exception occurs in “Stuff…,” we immediately “do something reasonable” and then “go on with life.”

- Descriptive string (if any) available as `e.getMessage()` for error messages and the like.
Exceptions: Checked vs. Unchecked

- The object thrown by `throw` command must be a subtype of `Throwable` (in `java.lang`).

- Java pre-declares several such subtypes, among them
  - `Error`, used for serious, unrecoverable errors;
  - `Exception`, intended for all other exceptions;
  - `RuntimeException`, a subtype of `Exception` intended mostly for programming errors too common to be worth declaring.

- Pre-declared exceptions are all subtypes of one of these.

- Any subtype of `Error` or `RuntimeException` is said to be `unchecked`.

- All other exception types are `checked`.
Unchecked Exceptions

• Intended for
  - Programmer errors: many library functions throw 
    IllegalArgumentException when one fails to meet a precondi-
    tion.
  - Errors detected by the basic Java system: e.g.,
    * Executing x.y when x is null,
    * Executing A[i] when i is out of bounds,
    * Executing (String) x when x turns out not to point to a String.
  - Certain catastrophic failures, such as running out of memory.

• May be thrown anywhere at any time with no special preparation.
Checked Exceptions

• Intended to indicate exceptional circumstances that are not necessarily programmer errors. Examples:
  - Attempting to open a file that does not exist.
  - Input or output errors on a file.
  - Receiving an interrupt.

• Every checked exception that can occur inside a method must either be handled by a `try` statement, or reported in the method's declaration.

• For example,
  ```java
  void myRead () throws IOException, InterruptedException { ... }
  ```

  means that `myRead (or something it calls) might throw IOException or InterruptedException.`

• Language Design: Why did Java make the following illegal?
  ```java
  class Parent {
    void f () { ... }
  }
  ```
  ```java
  class Child extends Parent {
    void f () throws IOException { ... }
  }
  ```
Good Practice

- Throw exceptions rather than using print statements and System.exit everywhere,

- ...because response to a problem may depend on the caller, not just method where problem arises.

- Nice to throw an exception when programmer violates preconditions.

- Particularly good idea to throw an exception rather than let bad input corrupt a data structure.

- Good idea to document when methods throw exceptions.

- To convey information about the cause of exceptional condition, put it into the exception rather than into some global variable:

```java
class MyBad extends Exception {
    public IntList errs;
    MyBad (IntList nums) { errs=nums; }
    try { ... }
    catch (MyBad e) {
        ... e.errs ...
    }
}
```