Defensive Programming
Which is more important

- When writing code for a moon rover communications sub-system.

A) Correctness
B) Robustness
Which is more important

- When writing code for a media player app (e.g., Pandora)

A) Correctness
B) Robustness
Which is more important

- When writing code for a high-frequency stock trading firm

A) Correctness
B) Robustness

Which is better?

A  
```java
public static void main(String [] args) {
    assert args.length >= 2 && args.length <= 3 :
        "This program takes 2 or 3 arguments";
    ...
}
```

B  
```java
public static void main(String [] args) {
    if (args.length < 2 || args.length > 3) {
        printUsage();
        return;
    }
    ...
}
```

C  Both are fine

D  Both are problematic
Sources of Invalid Inputs

- Where do bad inputs come from?
  - Bad command line parameters
  - Bad user input
  - Network (return bad data, corrupted in transmission)
  - Other code in project (your bugs)
  - File system configuration

- How should we address them?

- Assert
- Unexpected

- Infrequent
- Frequency
public static ArrayList<Integer> getDataFromFile(String filename) {

    ArrayList<Integer> parsedInts;
    assert (parsedInts = new ArrayList<Integer>()) != null;
    //
    try {
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        while (scanner.hasNext()) {
            parsedInts.add(scanner.nextInt());
        }
    }
    } catch (FileNotFoundException e) {
        return parsedInts;
    }

    return parsedInts;
}
What is wrong with this code

```java
public static int Sum(int [] array) {
    int sum = 0;
    try {
        for (int i = 0; true; i++) {
            sum += array[i];
        }
    } catch (ArrayIndexOutOfBoundsException e) {
        return sum;
    }
}
```
Exceptions: Usage rules

- only for exceptional situations
- handling errors in the appropriate place
  - but, handle in the smallest scope possible
- throw exceptions of the proper abstraction
- avoid empty catch blocks

"Code smell"
Kinds of Exceptions

```
+-----------------+              +-----------------+
| Throwable        |              | Checked         |
|                 +              +                +
| Error           | Checked       | Index out of bounds |
|                 +                +                +
| unchecked       |                | arithmetic      |
|                 +                +                +
| checked         |                |                  |
|                 +                +                +
| unchecked       |                |                  |
```

JVM

HW
Data Structure Integrity

E.g., For a doubly-linked list

```java
public class LinkedList {
    private LinkedList next;
    private LinkedList prev;

    public void checkIntegrity() {
        if (next != null) {
            assert next.prev == this :
                "prev pointer doesn't point to predecessor";
            next.checkIntegrity();
        }
    }
}
```
Barricade / Firewall

dirty input → unsafe sanitation checks → clean input → safe

→ assert

gracefully handle bad input
Command-line interface descriptions

usage: svn <subcommand> [options] [args]
        git diff [options] [<commit>] [--] [<path>...]

verbatim
<required component>
[optional component]
[multiple optional components...]
To Dos for Tuesday

- Read Ch. 11 (Power of Variable Names)
- Read Ch. 12 (Fundamental Data Types)
- Read Ch. 13.1 and 13.2 (Unusual Data Types)
  - 13.2 is optional given that it doesn’t really relate to Java

- Assignment description out today for extending your movie JSON parser