Debugging

- It is highly unlikely that you will write code that will work on the first go

- Bugs or errors
  - Syntax
    - Fixable if you learn to read compiler error messages
    - Lines number should take to erroneous section
  - Semantic
    - No easy fix
      - Use print statements to our advantage

Common Semantic Errors

- No re-initialization condition that will eventually halt the loop
  - For preferred over while due the nature of syntax
  - Condition starting out to be false initially
  - Wrong initialization value or end-test condition
  - So be careful of < vs. <=

Java Library

- Java provides a library or collection of useful programs
- A gold mine of well-tested code that can save you countless hours of development time
- This huge library information is provided in API - Application Programming Interface
**General Class Interface**

- A class' public **interface** (notice the four headings):
  - **Field Summary**: Has information about public variables
  - **Constructor and Method summaries**: Describe the public constructors and methods
  - **We will cover constructors soon…**
  - **Method Detail**: Provides detail on method inputs (parameter(s)) and output (return type) and some extra details
- The collection of all of a class' public fields, constructors, and methods constitute its "interface", that is, the public face that it shows the world.
- If method/variable is private, then it will be not a part of its interface

**Example Math Class**

- Notice the phrase **java.lang** at the top of the main panel above the word Math
- This means that the Math class is part of the core Java language and hence can be used directly
- Unlike the CPU class

**Math Class Interface**

- **Field Summary**: none
- **Constructor Summary**: has no public constructor
- **Methods Summary**: the methods implemented (all which are static)
- **Method Details**: e.g. abs(int a) Find the absolute value of a value

**Example CPU class**

- If a class is not part of java language i.e. **java.lang**, you'll see package name
- **What is a package?**
  - Basically it's a directory that has a collection of related classes
  - **E.g. CPU class – stamp.core.CPU**
    - Indicating that the CPU class program is stored in the folder stamp/core/ (within the Javelin folder)
- Since CPU is not part of Java Language we need to tell Java where to find it by saying
  - **import stamp.core.CPU**;
  - **Another way is to use the asterisk "wildcard character".**
    - **E.g. stamp.core.*;** which allows any classes in core folder to be accessed

**Psuedo-Code**

- **Psuedo-code** is a informal high-level description of an algorithm
- It uses the structural conventions of a programming language
  - Intended for human reading vs. machine reading
- Pseudo-code typically omits details that are not essential for human understanding of the algorithm
  - **E.g.** variable declarations, system-specific code and subroutines.
Psuedo-Code Rules

- Can use words such as while, if else-if
  - E.g. for 1 to n
- Do not specify data declarations or types
- Use Words that specify an action such as set, reset, increment, compute, calculate, add, sum, multiply, print, get input
- Use indentation for block of code i.e. {}

Example: Fibonacci sequences

- A Fibonacci sequence is an infinite list of integers
- The first two numbers are given
  - Usually (but not necessarily) these are 1 and 1
- Each subsequent number is the sum of the two preceding numbers:
  
 1 1 2 3 5 8 13 21 34 55 89 144 ...

- Let's write a program to compute sequence when we get to a number that is 1000 or bigger

Starting the Fibonacci sequence

- We need to initialize two numbers in the sequence
  - Set first to 1
  - Set second to 1
- We need to print these out:
  - Print first and second
- We need to compute and print the next number:
  - Set next to sum of first & second;
  - Print next

Taking the next step

- Now what?
  - Need to add second and next
    - set nextnext to sum of second & next
  - What if the sequence is too long
    - Do you want to make 100s of storage to hold each item
- The sequence so far is: first second next
  - Do I see a pattern emerging?
Preparing to make many steps

- We need to make these moves:
  - First second next
    - First second next
    - First second next
  - We can do it like this:
    - Set first to second
    - Set second to next
  - We can put these statements in a loop and do them as many times as we please

Complete Psuedocode

Set first to 1
Set second to 1
Print first and second
while next < 1000
  Set next to sum of first & second;
  print next
  Set first to second
  Set second to next

Scope

- Scope means the area of code in which an entity is known (or alive)
  - Mainly concerned with variables and methods
  - Which parts of the program can access them?
- Sometimes scope is explicitly designated with a keyword
  - private: known only within the class
  - public: known outside of (and within) the class
  - Note that Methods have explicit scope
- Other times it is implicitly designated by location

Implicit Scope: Method Parameters

- A method parameter is an “input variable”
- Scope: the method in which it is defined
- No other method can access (read/write) it

```java
public int absoluteValue(int n) {
    if (n < 0) {
        return -n;
    }
    else {
        return n;
    }
}
```
Implicit Scope: Local Variables

- A "local variable" is defined within a method body `{ }`
  - They are inherently private to the method in which they are defined
  - We don’t use public/private for local variables
- It may be defined in a block `{ }` within a method body
- **Scope**: point of declaration to end of closest enclosing block

Local Variable Scope: Example 1

```java
public static int isLarger(int x, int y){
    if (x > y) {
        int larger = x;
    } else {
        int larger = y;
    }
    return larger;
}
```

Local Variable Scope: Example 2

```java
void fibonacci(int limit) {
    int first = 1;
    int second = 1;
    while (first < limit) {
        System.out.print(first + " ");
        int next = first + second;
        first = second;
        second = next;
    }
    System.out.println( );
}
```

Global Variable

- Is variable declared outside of all methods within class
- Written at beginning of class
- **Syntax**
  ```java
  public static data-type variableName
  ```
  E.g. public static int count;
- **Scope**
  - **Accessible** (read and modified) by all methods within the class
  - Unless it is declared final
  - Ideally used as constants i.e. public static final int MAX;
  - To access global variable outside of the class it is declared in (as long as it is declared public)
  ```java
  Classname.globalVariableName
  ```
  E.g. CPU.pin10
**Code Tracing**

- A form of algorithm analysis in which you (the analyst) simulate computer operation i.e. you are evaluating the logic.
- You step through each line of code, determining what the execution of that line causes to happen.

**Call by Value**

- We call a method like this: `result = add(3, 5);`
- `int add(int number1, int number2)`
  ```java
  int sum = number1 + number2;
  return sum;
  ```

**Scope & Code Tracing Example**

- By default, copies of parameter are sent to a method
- `public static void main(){
  int x = 0;
  System.out.println("In main: x = " + x);
  foo(x);
  System.out.println("In main: x = " + x);
}
```

- `public static void foo(int x){
  System.out.println("In foo: x = " + x);
  x = 5;
  System.out.println("In foo: x = " + x);
}
```

- **Output:**
  - In main: x = 0
  - In foo: x = 0
  - In foo: x = 5
  - In main: x = 0

**Frame for method calls**

- Whenever a method is called, some memory is set aside to contain information related to the call.
  - Example: parameter values, current statement being executed etc.
  - This is called the frame for the method call.
- The frame is discarded when the method returns.
  - This implies parameters and local variables (variables declared within method) are discarded.
  - Hence, this explains the reasoning for scope rules.
- The frames are destroyed in the order in which they were created.
  - Follows first-in-last-out principle and the section of memory that allocates memory during method calls is known as stack.
Scope and for loops

- The for loop is a special case
  - You can declare variables in the for statement
  - The scope of those variables is the entire for loop
  - This is true even if the loop is not a block i.e. {}