6.178: Introduction to Software Engineering in Java

Lecture 2: 6.005 Getting Started: Git, Debugging, and Good Coding Practices

Git

Version Control & You

Version Control

- What is version control?
- Why use it?
- How do I use it?

Version Control Software

- Subversion
- Mercurial
- Git

About Git

- Created by Linus Torvalds
-
```
+
+    GIT - the stupid content tracker
+
+"git" can mean anything, depending on your mood.
+
+ - random three-letter combination that is pronounceable, and not
+   actually used by any common UNIX command.  The fact that it is a
+   mispronounciation of "get" may or may not be relevant.
+ - stupid. contemptible and despicable. simple. Take your pick from the
+   dictionary of slang.
+ - "global information tracker": you're in a good mood, and it actually
+   works for you. Angels sing, and a light suddenly fills the room.
+ - "goddamn idiotic truckload of sh*t": when it breaks
+
```

Why Git

- Has most popular support
- Has amazing ecosystem
  - See GitHub
- It’s decentralized

How to use Git

- Interface is primarily through the shell
  - there are some guis but nobody I know uses them (therefore they must suck right?)
- The shell can be scary
- GitHub adds cool features

Git Config

- Before using git, you should tell it who you are
  - git config --global user.name “Graeme Campbell”
  - git config --global user.email “graeme@mit.edu”
- Other cool things you can do to set up
- git init

Git Add

- git add <filename>
- git add -A
- git add *

Git rm

- git rm <filename>
- git rm *
- git rm -A

Git commit

- How to commit
    - git commit -m ‘commit message’
- Commit messages
- When to commit

External repositories

- Remotes:
    - git clone https://repo
    - git remote add <remote_name> https://repo
- git pull
- git push

Problems with Git

- Diagnose with git status
- Merge conflicts
- How to fix your issues
  - Nuclear option

Git is really complicated

- Why should I get good at it?
- Why bother with Git at all?

Git Exercises

The best way to learn git is to try it out. I’m going to walk you through some exercises here: https://github.mit.edu/6178-iap16/git-exercises/blob/master/README.md

Last lecture…

What we’ve learned so far...

- Types
- Variables
- Operators
- Control Flow: If, While, Do...while, For, Switch
- Variable Scope

Decision making example

```
if(n < 0) {
    System.out.println("I'm negative!");
} else if(n > 0) {
    System.out.println("I'm positive!");
} else {
    System.out.println("I'm lonely :(");
}
```

While Loops

```
while( STATEMENT ) {
    // do smart things
}
n = 3;
while( n > 0 ) {
    System.out.println(n); // Will print 3, 2, 1 and then exit
}
n = 3;
while( n % 2 != 0 ) {
    n *= 3; // Careful! Infinite loop, program will crash
}
```

Do...while loops

```
do {
    STATEMENTS
} while ( termination condition )
```

First do something, then check if you still need to do it.

```
int i = 0;
do { i++;
     System.out.println(i);
    } while ( i < 5 );
```

For Loops

```
for(initialization;condition;update){
    statements
}

for(int i=0; i < 3; i++){
    System.out.println(“Rule# “ + i);
}
```

Switch statement - Decisions, decisions...

We use this when we have a decision task with multiple cases and, instead of using if...else if a lot of times, we can use switch.

```
switch(variable) {
    case CASE1 : /* ... */;
    case CASE2: /* ... */;
    case CASEn: /* ... */;
    default: /* ... */;
}
```

Today!

- Good Coding Practices
- Debugging
- Intro to Data Structures

Good Coding Practice

#1: Don’t Repeat Yourself! (DRY)

```
public static int dayOfSpringSemester(int month, int dayOfMonth, int year) {
    if (month == 2) {
        dayOfMonth += 31;
    } else if (month == 3) {
        dayOfMonth += 31 + 28;
    } else if (month == 4) {
        dayOfMonth += 31 + 28 + 31;
    } else if (month == 5) {
        dayOfMonth += 31 + 28 + 31 + 30;
    }
    return dayOfMonth;
} // BAD! What if we predefined some constants for each month’s number of days? What if we used a
for loop? Would’ve definitely had a cleaner and shorter code.
```

Good Coding Practice

#2: Comment where needed! Remember, comments are for humans, both you and other developers who’ll try to work with your code.

```
/**
 * Return the Fibonacci numbers until n
 * @param n the number of Fibonacci numbers; requires n >= 0.
 * @return the Fibonacci sequence starting at 1 and with size n.
 * For example, Fibo(6)=[1, 1, 2, 3, 5, 8].
 */
public static List<Integer> FibonacciSequence(int n) {
...
}
```

Good Coding Practice

#3: Use good, self-explanatory names for variables, methods, classes, etc…

```
class thing...
String a1;
int a2;
double b;  //BAD!!
```

```
class Names...
String firstName;
String lastName;
int temperature; //GOOD
```

Good Coding Practice

#4: Use indentation and spaces!

```
public static void main(String[] args){

int x=5; x=x*x;

if(x>20) System.out.println(x+“>20.”);double y=6.178;}
```

Ctrl-shift-F to auto-format the file

Good Coding Practice

#4: Use indentation and spaces!

```
public static void main(String[] args){
    int x=5;
    x=x*x;
    if(x>20)
        System.out.println(x+“>20.”);
    double y=6.178;
}
```

Much Better! :)

Debugging

- System.out.println() is your best friend!
- Think about what variables change after every executed line.
- The computer isn’t stupid. It just does what you tell it.
- Eclipse has a good debug tool too.

Let’s debug!

A quick word on the enum type

It’s your own type!

```
public enum Day {
    MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

Day day = Day.FRIDAY;
```

enum used with switch

```
Day day = Day.FRIDAY;
switch(day){
    case FRIDAY: System.out.println("It's Friday, Friday!");
                 break;
    case SATURDAY: System.out.println("Sleeping in on Saturday");
                   break;
    case SUNDAY: System.out.println("Sunday...gotta pset");
                 break;
     default: System.out.println(“Not a weekend...Much hosedness”);
                 break;
}
```

Let’s learn some data structures!

Why use them?

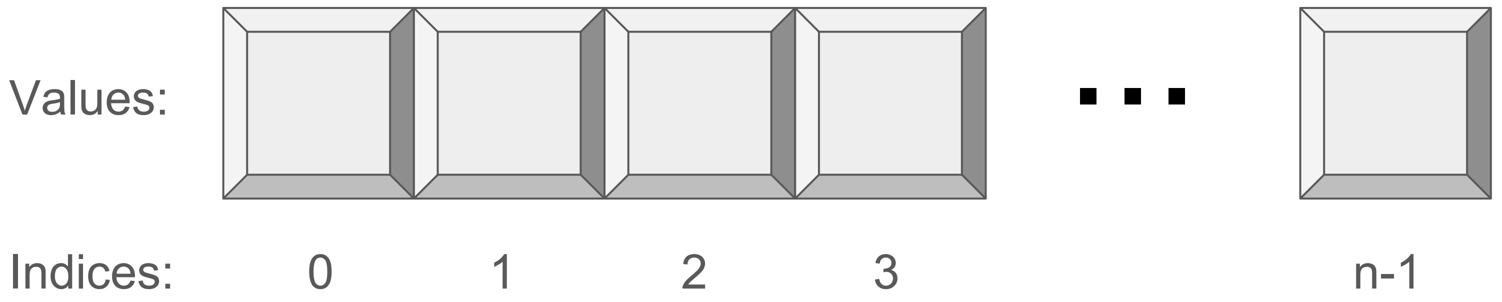
Because there are certain properties of data structures that help with algorithmic processing. Data structure is a particular way of storing and organizing information in a computer so that it can be retrieved and used most productively.

Data Structures

- **Arrays**
- Vector
- Linked List
- Set
- HashMap
- Stack
- Queue

Arrays

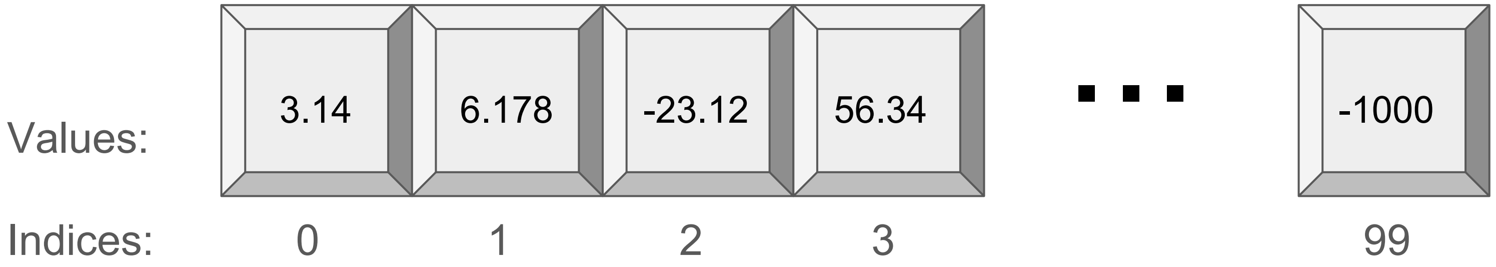
- An array is an indexed list of values.
- You can make an array of any type – int,double,String,etc...
- All elements of array must have the same type.
- Remember our boxes of variables! (foo and booHoo?)



Arrays

- Type[] arrayName = new Type[INT_SIZE];
- Arrays are of a fixed size. (Coming soon: ArrayList)
- Zero-indexing

```
double[] myNumbers = new double[100];
```



Arrays

```
int[] allTheNumbers = new int[1000]; // using an int

int numberOfNumbers = 1500;
int[] allNums = new int[numberOfNumbers]; // using a variable

int[] moreNums = new int[24*33 - 12]; // using an expression
```

Use any way of declaration!

Arrays

… Or use your own list of values:

```
Type[] array = {/* LIST OF VALUES */};
```

```
String[] myMagicalArray= {"JAVA", "MIT", “Cauliflower”};
```

… Or have a multidimensional grid!

```
int[][] my2DArray = new int[10][10];
```

Dynamic Arrays - aka Lists

- Like arrays, only dynamic
- You don’t have to specify the size of the list when you initialize it; it dynamically allocates space for you as you add more elements
- They have their own methods like size, add, addAll, remove, toArray, indexOf that you can find in the Java documentation
- They come in a multitude of shapes and forms: ArrayList, LinkedList, Stack, etc… We’ll mostly use ArrayLists

ArrayList

Example:

```
String myName = “Andreea”;
String yourName = “Kate”;
List<String> names = new ArrayList<>();
names.add(myName);
System.out.println(names.size()); // Prints 1
names.add(yourName);
System.out.println(names.size()); // Prints 2
String myTempName = (String) names.get(0); // myTempName = “Andreea”
String yourTempName = (String) names.get(1); // yourTempName = “Kate”
```

Data Structures

- Arrays
- **Vector**
- Linked List
- Set
- HashMap
- Stack
- Queue

Vector vs. ArrayList

- A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.
- A Vector is thread-safe (but you will care more about this in 6.005)
- Works the same way as an ArrayList! (see Java documentation)

Coding Exercise

Let’s play around with these!

```
git clone https://github.mit.edu/6178-iap16/<kerberos>_lec2
```