Working as part of a group

see McCarthy, *Dynamics of Software Development*

- **establish a shared vision**
  - what was/is Hyperwag? what can we add?
  - harmonious sense of purpose

- **develop a creative environment**
  - the more ideas the better, ideas are infectious
  - don’t flip the BOZO bit

- **scout the future**
  - what’s coming, what’s the next project
  - what new technologies will affect this project
Scheduling/Slipping

- **McCarthey page 50, Group Psyche, TEAM=SOFTWARE**
  - anything you need to know about a team can be discovered by examining the software and vice versa
  - leadership is interpersonal choreography
  - greatness results from ministrations to group psyche which is an “abstract average of individual psyches”
  - mediocrity results from neglect of group psyche
- **Slipping a schedule has no moral dimension (pp 124-145)**
  - no failure, no blame, inevitable consequence of complexity
  - don’t hide from problems
  - build from the slip, don’t destroy
  - hit the next milestone, even if redefined (”vegetate”)

Software Design
Towards being a hacker

- See the hacker-faq (cps 108 web page)
  - Hackers solve problems and build things, and they believe in freedom and voluntary mutual help. To be accepted as a hacker, you have to behave as though you have this kind of attitude yourself. And to behave as though you have the attitude, you have to really believe the attitude.

- The world is full of fascinating problems
  - no one should have to solve the same problem twice
  - boredom and drudgery are evil
  - freedom is good
  - attitude is no substitute for competence

*You may not work to get reputation, but the reputation is a real payment with consequences if you do the job well.*
Aside: ethics of software

- What is intellectual property, why is it important?
  - what about FSF, GPL, copy-left, open source, ...
  - what about money
  - what about monopolies

- What does it mean to act ethically and responsibly?
  - What is the Unix philosophy? What about protection? What about copying? What about stealing? What about borrowing?
  - No harm, no foul? Is this a legitimate philosophy?

- The future belongs to software developers/entrepreneurs
  - what can we do to ensure the world’s a good place to be?
Software Design

See Alan Cooper, *The Essentials of User Interface Design*

- who designs the software?

- Implementation is view of software developer, user’s view is mental model, software designer has to bridge this gap
  
  Example: copy/move files in a Windows/Mac environment, what’s the difference in dragging a file/folder between two folders on the same device and dragging between devices, e.g., c: to a:? Is this a problem? To whom?

- Implications in Hyperwag? What’s an appointment? A day?
Comfort with technology and mathematics

- “Show me all the first year students who live in Pegram and in Brown”
  - what does “and” mean here? Does the average user understand Boolean? Does the average programmer understand Boolean? Recursion? Threads? Queues?
- How you solve a problem in your program isn’t (necessarily) how the user solves the problem, keep these distinctions clear.
- “Saying that someone is ‘computer literate’ is really a euphemism meaning he has been indoctrinated and trained in the irrational and counter-intuitive way that file systems work, and once you have been properly subverted into thinking like a computer nerd, the obvious ridiculousness of the way the file system presents itself to the user doesn’t seem so foolish.”
From C++ to Java

• **Java history:** Oak, toaster-ovens, internet language, panacea
  - Not really a standard language like C++
  - Arguably proprietary (and arguably not)
  - Precursor to C#?

• **What it is**
  - O-O language, not a hybrid (like C++)
  - Compiled to byte-code, executed on JVM
  - Byte-code is “highly-portable”, write once run “anywhere”
  - Simple, object-oriented, portable, interpreted, robust, secure, architecture-neutral, distributed, dynamic, multithreaded, high performance
Java on one slide

- **All objects allocated on heap, via new, garbage collected**
  - Primitive types like int, double, boolean exempt
    - Everything else subclasses Object
  - All variables (non-primitive) are pointers aka references
    - Can we compare pointers for equality? Is this a problem?
- **No free functions, everything in a class, inheritance by default**
  - Functions and classes can be *final*, not inheritable from
  - Static functions like Math.sqrt are like free functions
  - Local variables must be assigned to, instance variables all initialized by default to 0, null
- **Containers contain only non-primitive types**
  - Conversion between int and Integer can be ugly
  - Use ArrayList and HashMap instead of Vector, Hashtable
Java on another slide

- **Public class Foo must be in a file Foo.java**
  - Compiled into Java bytecodes, stored in Foo.class
  - Bytecodes executed inside a JVM: Java Virtual Machine
  - JVM is architecture specific, often relies on native/C code
- **Helper/non-public classes can be in same file**
  - Keep related/cohesive concepts together
  - Don’t go overboard

- **Execution starts with a static main function**
  - Any class can have such a function, class invoked specifically via `java Foo` (runs Foo.main)
- **The environment is important and essential**
  - You need to understand CLASSPATH to leverage Java
From STL to Java

- In STL an iterator is a concept, there are refinements
  - Input, output, forward, bidirectional, random access
    - A forward iterator is an input iterator and an output iterator
    - The iterator may be immutable (or const)---read only
  - Refinements not implemented by inheritance, but by design, contract, and subsequently implementation
    - What happens if you try to implement an STL iterator?

- In Java *Iterator* is an interface (like a base class), similar to Tapestry iterators
  - Collection(s) are required to have iterators, these are used in some operations like max, min, construct vector, ...
  - Related to STL as algorithm glue, but very different
public void print() {
    Iterator allKeys = myMap.keySet().iterator(); // words

    while (allKeys.hasNext()) {
        String key = (String) allKeys.next();
        System.out.print(key + "\t");
        Iterator lines = ((Set) myMap.get(key)).iterator();
        while (lines.hasNext()) {
            System.out.print((Integer) lines.next() + " ");
        }
        System.out.println();
    }
}

- **Differences between Java and Tapestry in practice?**
  - Must store current element since `next()` does two things
  - Must cast since Collections store Objects
Java inheritance

- By default every class can be a base/parent class, every method is polymorphic. To inherit use `extends` keyword
  - Can change with final keyword (similar to const, but not)
  - A class can extend only one baseclass (but see interfaces)
  - Public, protected, private similar to C++, what’s not?

- A class can be an abstract class, `public abstract class Foo`
  - Can’t instantiate (no new Foo()), but can extend
  - A method can be abstract, like pure virtual in C++

- A class implements any number of interfaces
  - Like ABC, but function prototypes only, no state
  - Subclass must implement all methods of interface
Interfaces, Comparator, Inner classes

- The `java.util.Comparator` interface is used in sorting
  - Different from the `java.lang.Comparable` interface?
  - What must be implemented?

- Suppose we want to change sort in `WordLinesHashProxy`
  - If we change `keySet` to `entrySet` what’s in `ArrayList`?
  - Program compiles/doesn’t run sorting `Map.Entry` objects
    - How is this different from C++ behavior?

- How can we sort by size-of set while still sorting strings?
  - Use anonymous inner class that implements `Comparable`
  - Syntax is strange: create new interface
  - Access local variables, but some rules on parameters
Modules and Packages

- **Java code/modules organized into packages**
  - C++ has namespaces, not often used (broken in g++)
  - Java uses packages: corresponds to directory hierarchy
  - We’re using the default package (no name) later we’ll use packages
  - java.util, java.lang, java.io, ... are all packages

- **The import statement at the beginning of a program doesn’t work like #include, it tells the Java compiler where to look to resolve names**
  - Differences from #include/pre-processor?
Access modifiers aka public/private

- Classes that are related should be grouped together, may even share access to otherwise private methods/instance variables
  - A package corresponds to a directory just as class corresponds to file
  - Each method/instance variable must be declared as
    - public, accessible to all client code (other packages)
    - private, accessible only to class itself
    - protected, accessible to subclasses and package classes
    - (nothing), accessible to classes in same package

- One public class per file
  - Can have helper classes in the file, these will have package scope
  - Can have nested classes, very useful, also in C++
Class and class design in Java

- **Classes can be nested in Java**
  - Inner class has access to an object’s internal state
  - Static Inner class doesn’t belong to an object
    - Similar to use of Node we’ve seen in C++ programs
    - Why should Node be nested, private?
  - We’ll see anonymous inner classes later

- **Common constructor code, one constructor calls another**
  - `this(...)`
    - First code in a constructor, contrast to solution in C++
- **Call super/parent class constructor**
  - `super(...)`
    - Contrast to C++, must name class
What’s slow in Java?

- **Strings are notoriously slow**
  - Strings are immutable, contrast to StringBuffer
  - String concatenation isn’t, calls to StringBuffer

```java
String s = "hello";
s = s + " world "
s = (new StringBuffer()).append(s).append("hello").toString();
```

- Contrast to sequence of calls, how many calls to new?

```java
String s = "hello";
s += " world ";
s += " goodbye";
```

- **I/O is supposed to be slow**
  - My experiments show input is ok, output is very slow
  - Largely due to conversion of characters to bytes (unicode)
Proxy/Adapter patterns help

- **Proxy forces:** control access to an object, need more versatile reference than raw pointer, replace raw pointer with smarter substitute
  - *Solution:* stand in acts like object, forwards requests
  - Proxy used as the object for which it stands in
  - Transparent to client program
  - Often used for remote objects, lazy creation, protection
  - Used in C++ strings for copy-on-write/memory sharing
- **Adapter forces:** use class, but interface not what we need
  - *Solution:* clients call wrapper/adapter, requests forwarded
  - Wrapper can alter behavior, add to it, ...
- **Proxy provides same interface to what it adapts, Adapter changes interfaces**
In `WordLines.java` a set of Integer objects tracks line numbers

- What is the problem, why unavoidable with `Collection`?
- What is a solution or steps towards a solution?

A string is created for every word, is there an alternative?

- Store all characters in a file in memory
- String is an offset and length that “points” to chars
  - Proxy(0,1), Proxy(2,4), ...

More optimizations: touch every char once, don’t visit again when hashing (visited once to find words)
What does a String Proxy need?

- **Must implement the Comparable interface**
  - What is contract of `compareTo(…)` function?
  - What is formal parameter?

- **Must implement `equals(..)`**
  - Different from `compareTo(..)`? Why?

- **Must implement `hashCode()`, identical to `String`?**
  - How can we ensure same hashing? Is this needed?

- **Must implement `toString(..)`**
  - When used, alternatives? Different I/O?