CSC207H: Software Design

Wael Aboelsaadat

wael@cs.toronto.edu

http://ccnet.utoronto.ca/20075/csc207h1y/

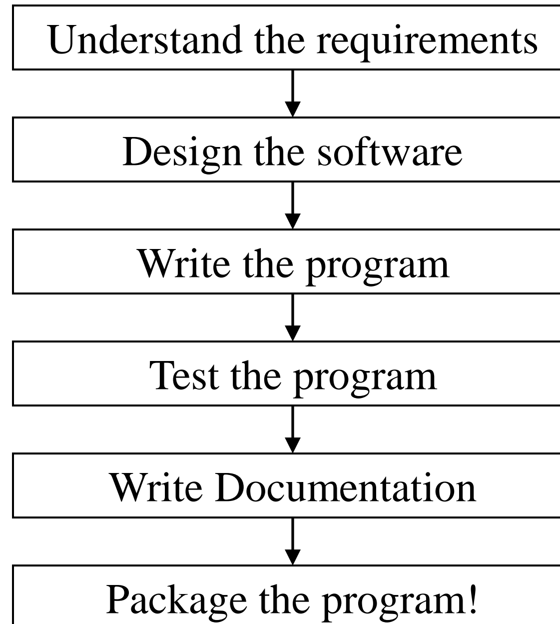
Office: BA 4261

Office hours: R 5-7

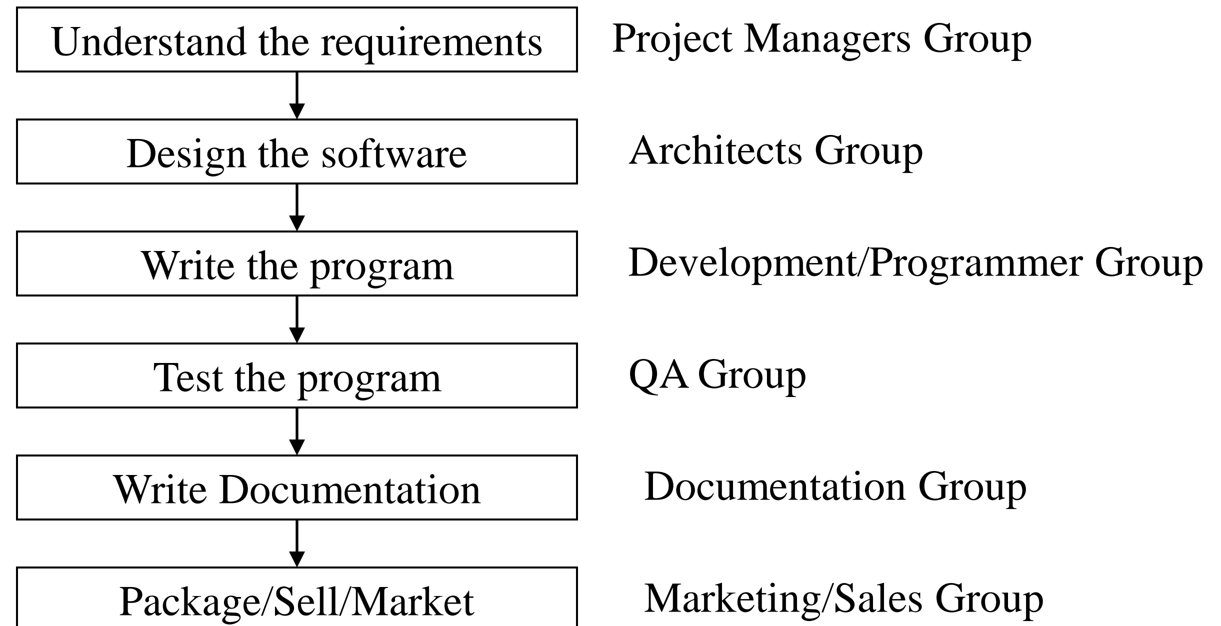
Acknowledgement: These slides are based on material by Prof. Karen Reid

Software house: what happens inside?

Software house: what happens inside?



Software house: what happens inside?



Tools in a Software House

Tools in a Software House

- Programming Language(s)!
- Scripting Language(s)
- Integrated Development Environment (IDE) App
- Profiling Tools
- Version Control App
- Quality Assurance Framework
- Software Build Management Framework
- Requirements/Feature Tracking App
- Variance Tracking App
- Architecture Tools

Tools in a Software House

Tool	Used By
Programming Language	
Scripting Language	
IDE	
Profiling tools	
Version Control App	
Quality Assurance Framework	
Software Build Management Framework	
Requirements/Feature Tracking App	
Variance Tracking App	
Architecture Tools	

Tools in a Software House

Tool	Used By
Programming Language	Programmers
Scripting Language	Programmers
IDE	Programmers
Profiling tools	Programmers, QA
Version Control App	Programmers, QA
Quality Assurance Framework	Programmers, QA
Software Build Management Framework	Programmers
Requirements/Feature Tracking App	Managers, QA, Programmers, Architects
Variance Tracking App	Programmers, QA, Managers
Architecture Tools	Architects, Programmers

Course Content

- Software Tools:
  - Version control
  - Build management
  - Testing
- Software Design:
  - object-oriented programming
  - program design (design patterns, testing patterns)
  - Reflection, refactoring
- Automation:
  - Python programming language
  - Shell scripting

Version Control App

Problem 1: working solo

- How do you keep track of changes to your program?

- Option 1: Don't bother
  - Hope you get it right the first time
  - Hope you can remember what changes you made if you didn't
  - (You probably won't get it right)
  - (Or remember)
  - (You *will* end up rewriting code)
  - (I do!)

Working solo (cont.)

- Option 2: Periodically save “backups”
  - Save snapshots of your program in another directory or under a different name
    - *E.g.* Main.1.java, Main.2.java
    - Or save it in a directory by date
  - Problems:
    - Totally ad hoc
    - Only the programmer knows how to interpret the names
    - Hard to pick a version to go back to
    - Prone to error
    - No tools to help you

Problem 2: working in a team

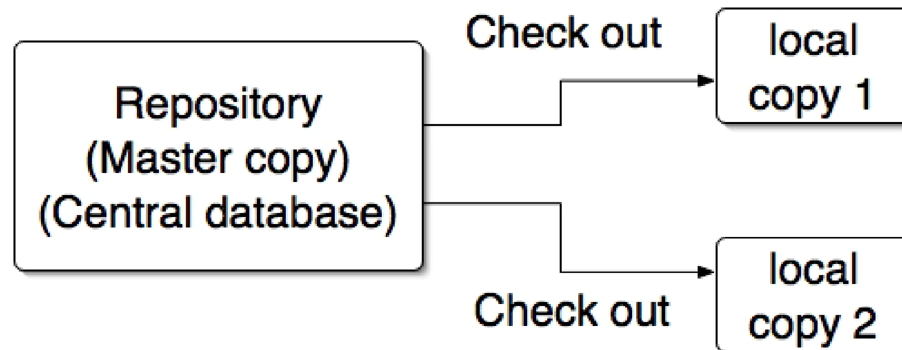
- How do you coordinate who has the authority to change a file?
- Worry about it after the fact
  - “Hey, why is this broken?”
  - “My changes got overwritten!”
  - “You weren't supposed to change that file.”
- Exchange email
  - “Okay, I'm going to work on A.java, so don't touch it.”

Problem 3: moving around

- After a hard day of work in the lab, you want to go home and do some work at home in the evening.
- How do you know which files to copy to your home machine?
  - Copy everything
    - potentially slow
    - might overwrite something you did at home, but forgot to copy from home to school
- Try to remember what changed
  - highly likely to get it wrong

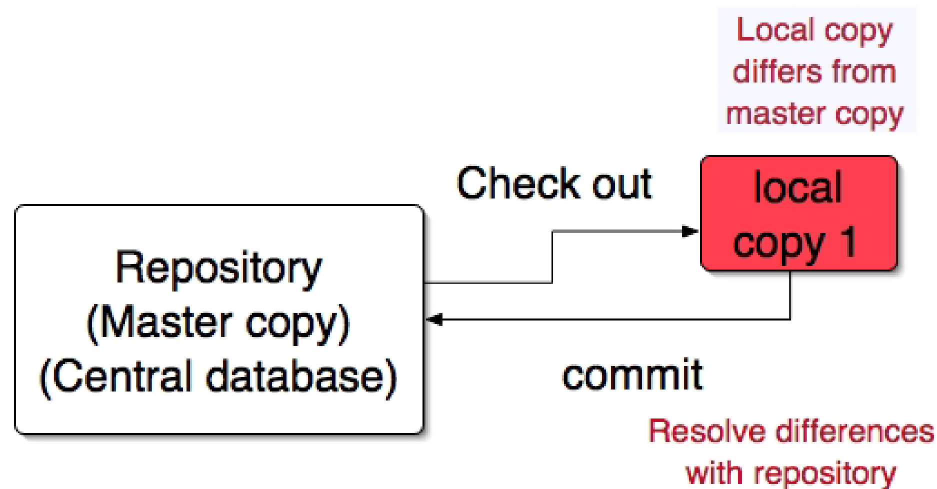
Solution: version control

- Keep code in a central location ("repository")
  - This is the master copy
  - Never directly modify this directory
- Create a local copy of the repository in your account at school, on your machine at home, on your laptop...



Commit

- When the local copy changes, “commit” the changes to the repository



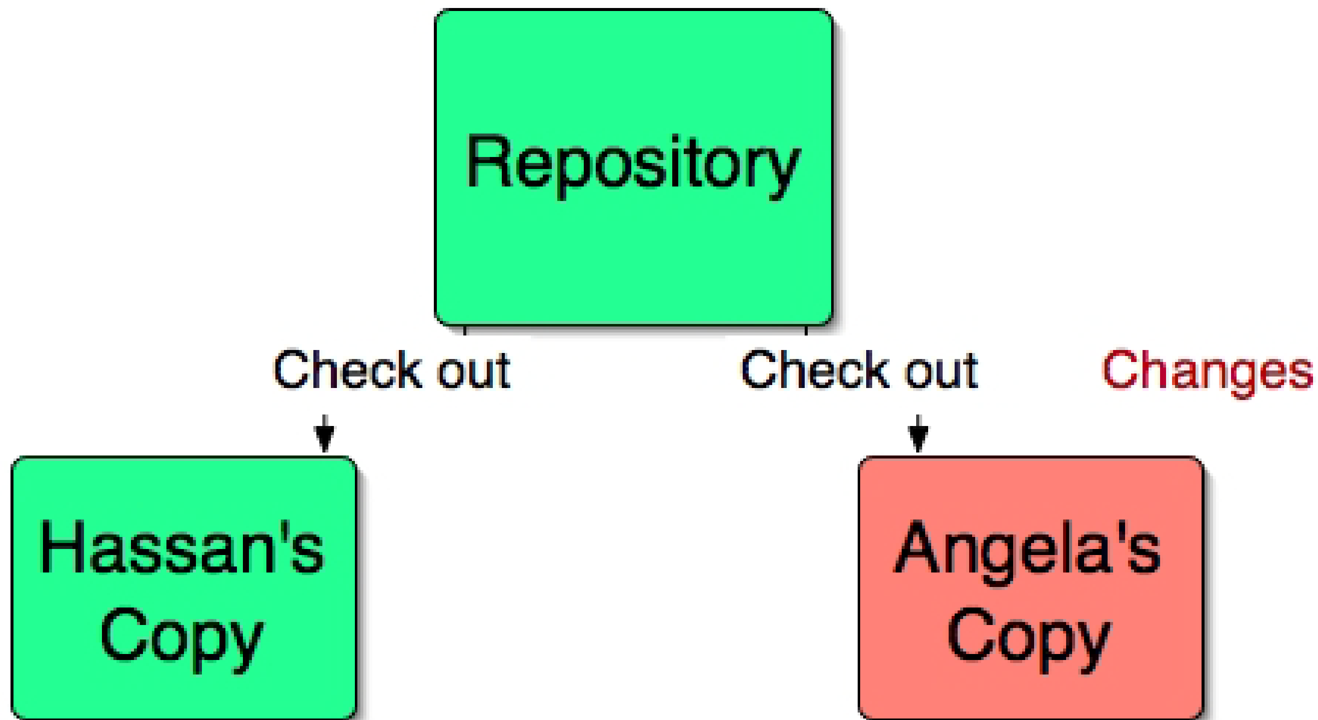
Solving problem 1

- When you get something working, commit the changes
- Tools allow you to revert to a previous version
- Write good log messages so that you don't have to remember what changed in each version

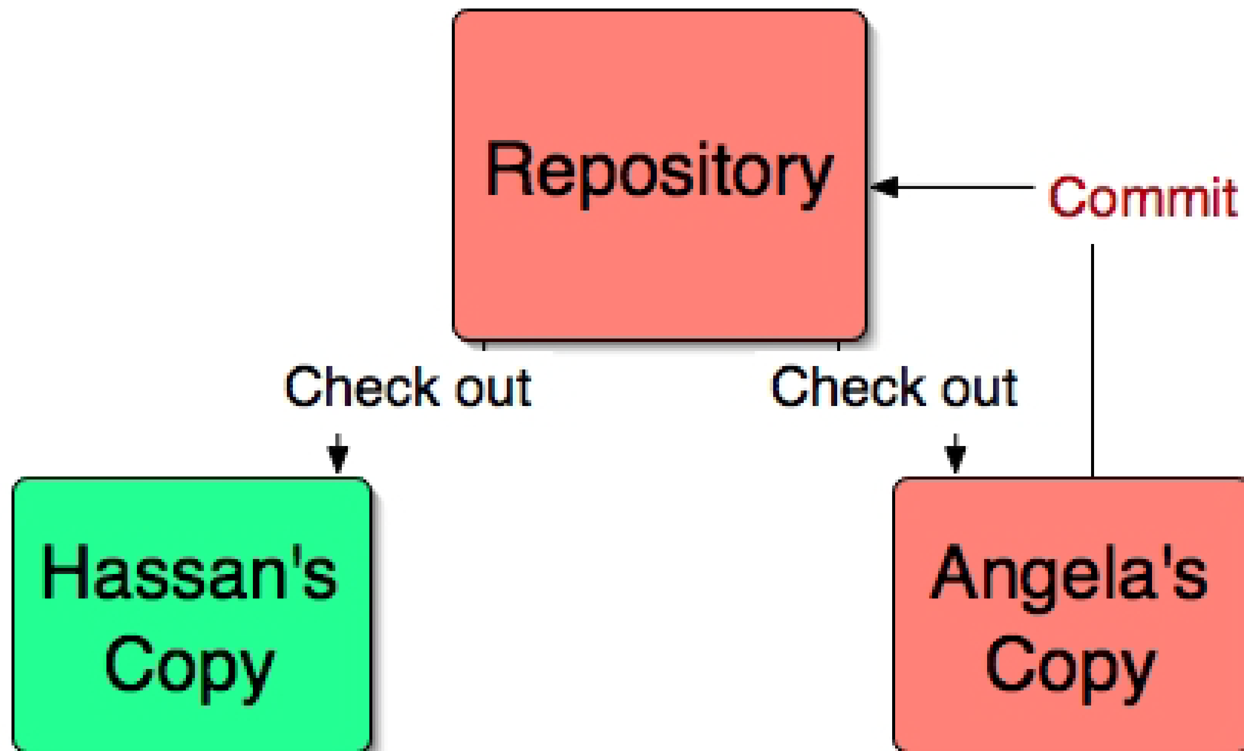
Managing concurrency

- What if two (or more) people want to edit the same file at the same time?
- Option 1: prevent it
  - Only allow one writeable copy of the file
  - Pessimistic concurrency
  - Microsoft Visual SourceSafe
- Option 2: patch up afterward
  - Optimistic concurrency
  - "Easier to get forgiveness than permission"
  - CVS, Perforce, Subversion

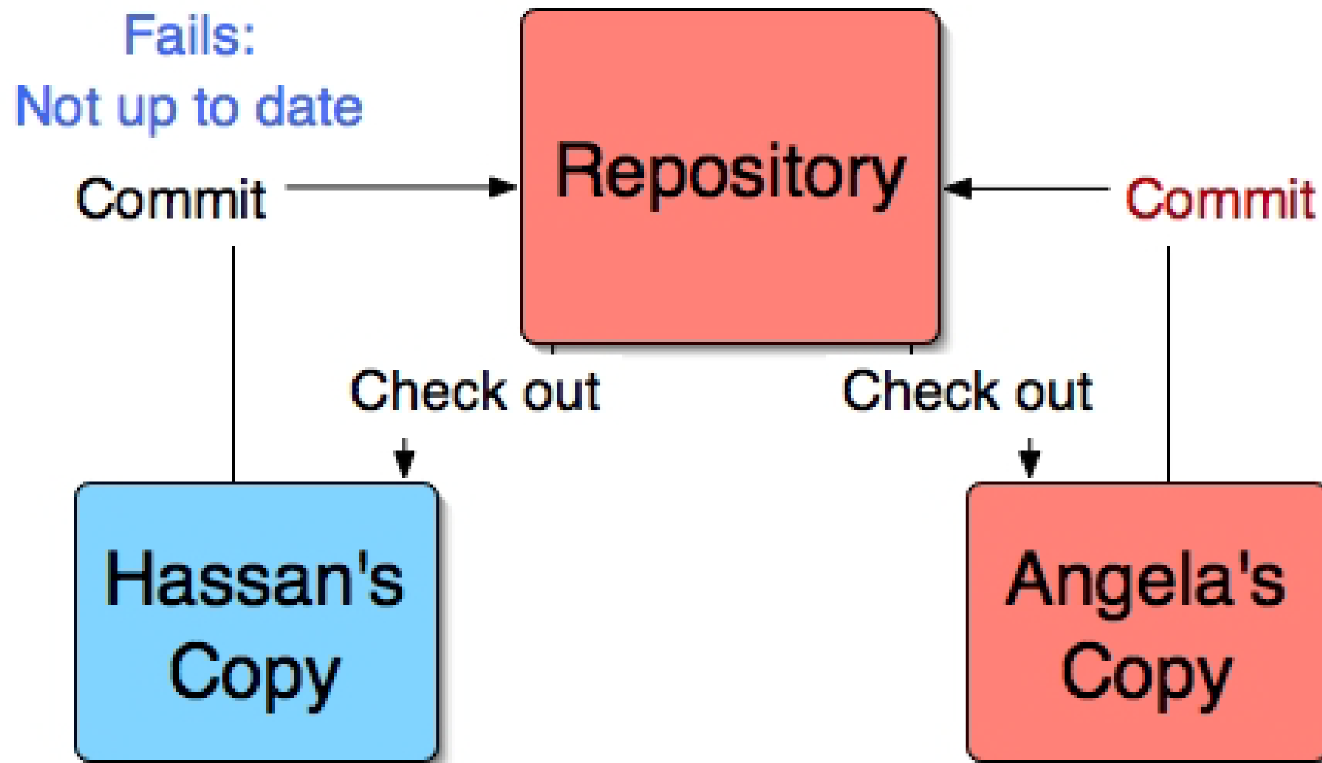
Optimistic concurrency: example



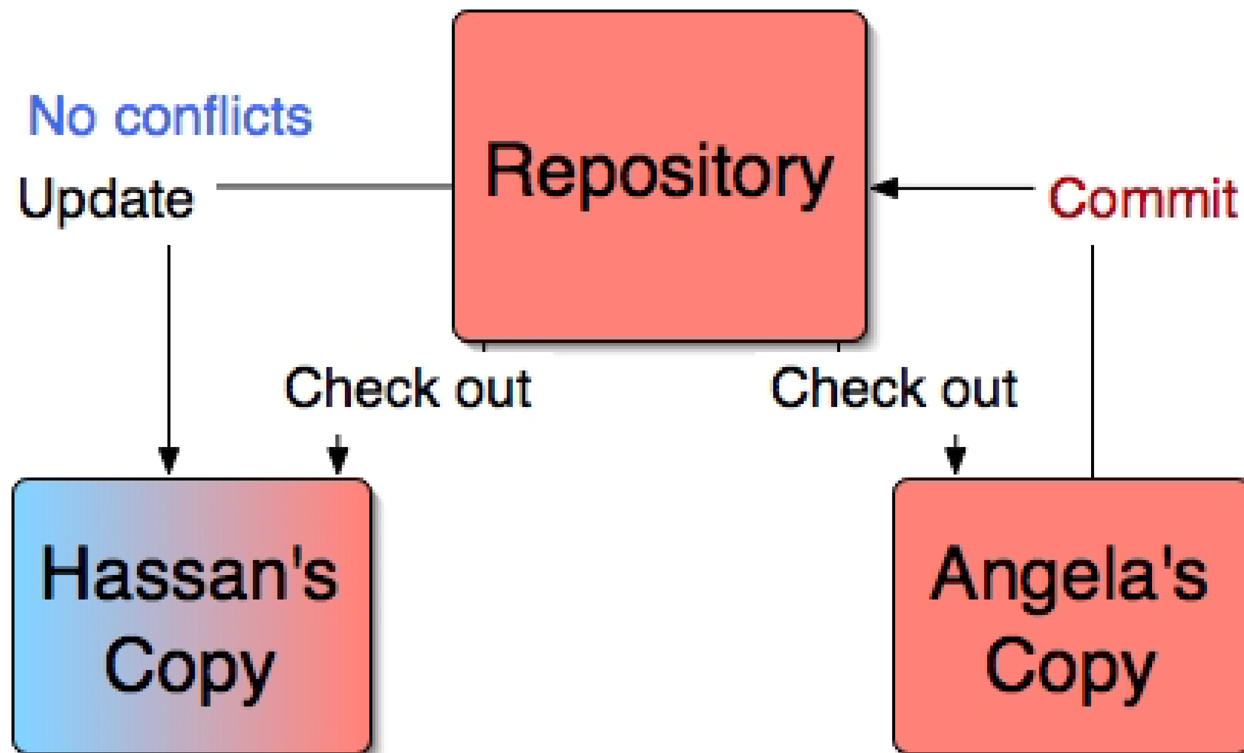
Angela commits changes



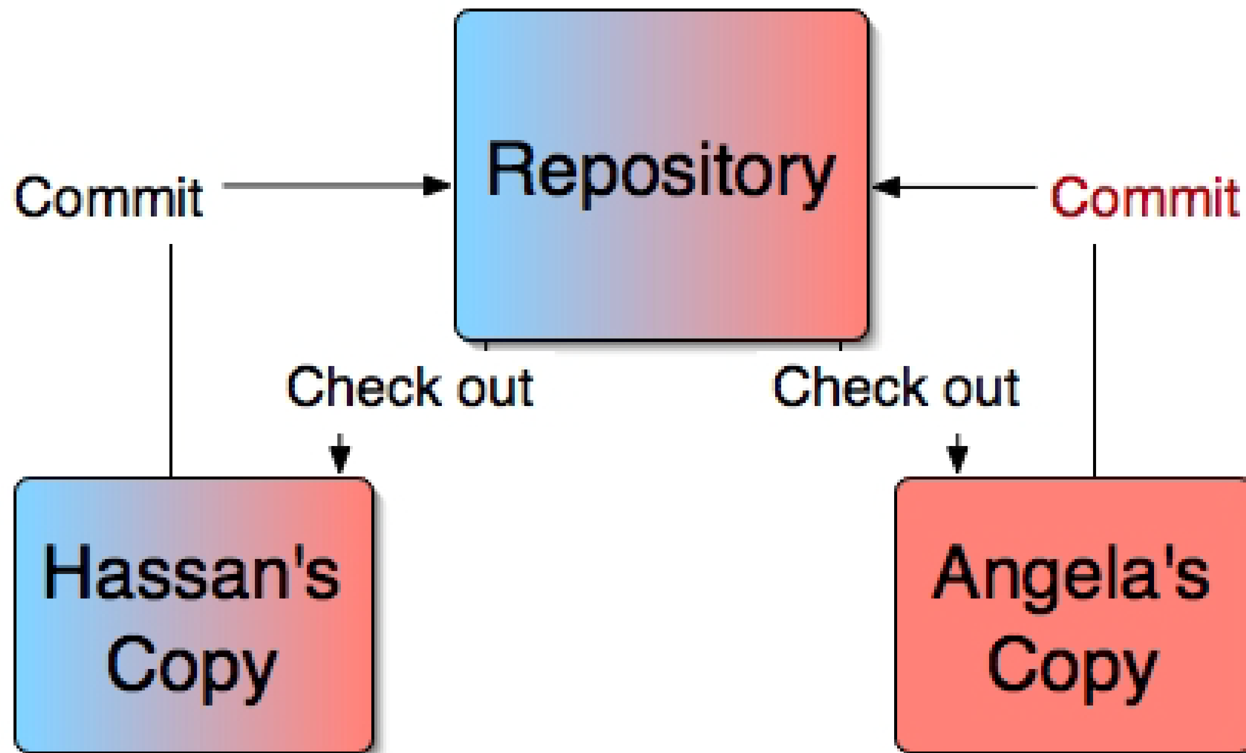
Hassan tries to commit changes



Hassan updates his version



Hassan now commits



Conflict

- To resolve a conflict, Hassan must manually fix the conflict before he commits

```
<<<<<<< A.java
   Iterator i = changedGroups.iterator();
   while (i.hasNext()) {
      if (userNotInGroup(currentUser, (Group) i.next())) {
         addUserToGroup(currentUser, (Group) i.next()));
      }
   }
=======
   Iterator i = changedGroups.iterator();
   while (i.hasNext()) {
      Group g = (Group)i.next();
      if (userNotInGroup(currentUser, g)) {
         addUserToGroup(currentUser, g));
      }
   }
>>>>>>> 1.3
```

Storage scheme

- Storing entire copy of a file on every change would require an enormous amount of disk space
- Version control systems store incremental differences
- The incremental differences allow the system to reconstruct previous versions

CVS

- “Concurrent Version System”
- Background information:
  - Each project is called a *module*
  - Command-line tool on Unix and Windows
  - Built on top of older system called RCS
  - Several graphical and web interfaces
  - The Eclipse IDE has a CVS feature
- We will create one repository for each of you
- **You will use CVS to submit work in this course**

Common CVS commands

- cvs checkout [module] Get initial copy
  - Do this every time you want to get a new copy of a module
  - CVS path to repo on CDF: /u/csc207h/summer/pub/repo/*CDFid*
  - Tip for exercises: after you commit the version you want us to mark, check out a fresh copy and test it on CDF
- cvs add [filenames] Add new files
  - Notifies CVS that you want it to track the new files
  - add must almost always be followed by a commit because add doesn't actually modify the repository

Common CVS commands

- cvs update [filenames] Synchronize with repository
  - Copies stuff from the master repository to your local copy
  - Any commits made by another person or commits done by you from another local copy will be updated in your local copy
  - Does not change the repository
  - Watch the messages closely

- cvs commit [filenames] Commit local changes
  - Copy changes to the repository
  - Will only be allowed if local copy is up to date

Common CVS commands

- cvs remove [filenames] Remove files from repository
  - Must first remove the file from the local copy
  - It is not easy to remove directories
  - Need to commit, just like for add

- cvs diff [filenames] Show diffs between local
- copy and repository
  - Handy to see what changed between versions

- cvs log [filenames] Show history of files
  - Shows log message, timestamps and who made the revisions

Keyword expansion

- CVS replaces strings like $Keyword:$ with associated value during check-in
- $Author: mcraig $
- $Date: 2006/06/06 13:59:59 $
- Always in Universal Time
  - Don't let cvs log fool you
- $Revision: 1.1.1.1 $

Keyword expansion

- Often embed these values in strings, rather than in comments

```
class Something implements Versioned {
  public static final String VERSION = "$Revision: 1.1.1.1 $";

  public String getVersion() {
    return VERSION.split(' ')[1].trim();
  }
}
```

File types

- CVS treats files as text by default
  - Tries to perform keyword expansion
  - Tries to translate Unix and Windows line endings
- Often don't want this
  - *E.g.* image files
  - Add with `cvs add -kb filename`
    - kb means "binary"
- Manual explains how to set file type after the fact

What to check in?

- Check in anything you created by hand
  - Your own little test programs
  - And their expected output
  - Readme files, notes, build logs, etc.

- Don't check in files that are automatically created from others. *E.g.* .class files
  - Uses disk space and bandwidth for no reason: they get overwritten the next time you compile

When to check in?

- Version control is not a backup system
  - Your computer should have one of those
  - It's not a synchronizing tool, either
- Don't check in just because you're taking a break to surf the web
- *DO* check in files when they are stable
  - *E.g.* after adding a new feature, or passing another test
- Or when you have to move from location to location
  - *E.g.* from home to school or vice versa

Other Version Control Systems?

Revisions of '/Text-TagTemplate/TagTemplate.pm'

Revision	Tags	Date	Author	Comment
***1.14**		**6/7/05 6:39 PM**	**matisse**	**Added new methods: tag_start[...]**
1.13		5/11/04 12:38 AM	matisse	Bumped version to 1.81
1.12		5/11/04 12:34 AM	matisse	Supressed warnings when replacement strin
1.11		5/10/04 7:58 PM	matisse	Increased version number to 1.8
1.10	effective_date_override_:	12/2/03 8:41 AM	matisse	Changed version number to 1.7 because CP/
1.9		11/20/03 2:27 PM	matisse	Fixed VERSION in perldoc. (Set to 1.6.1)

Text Compare

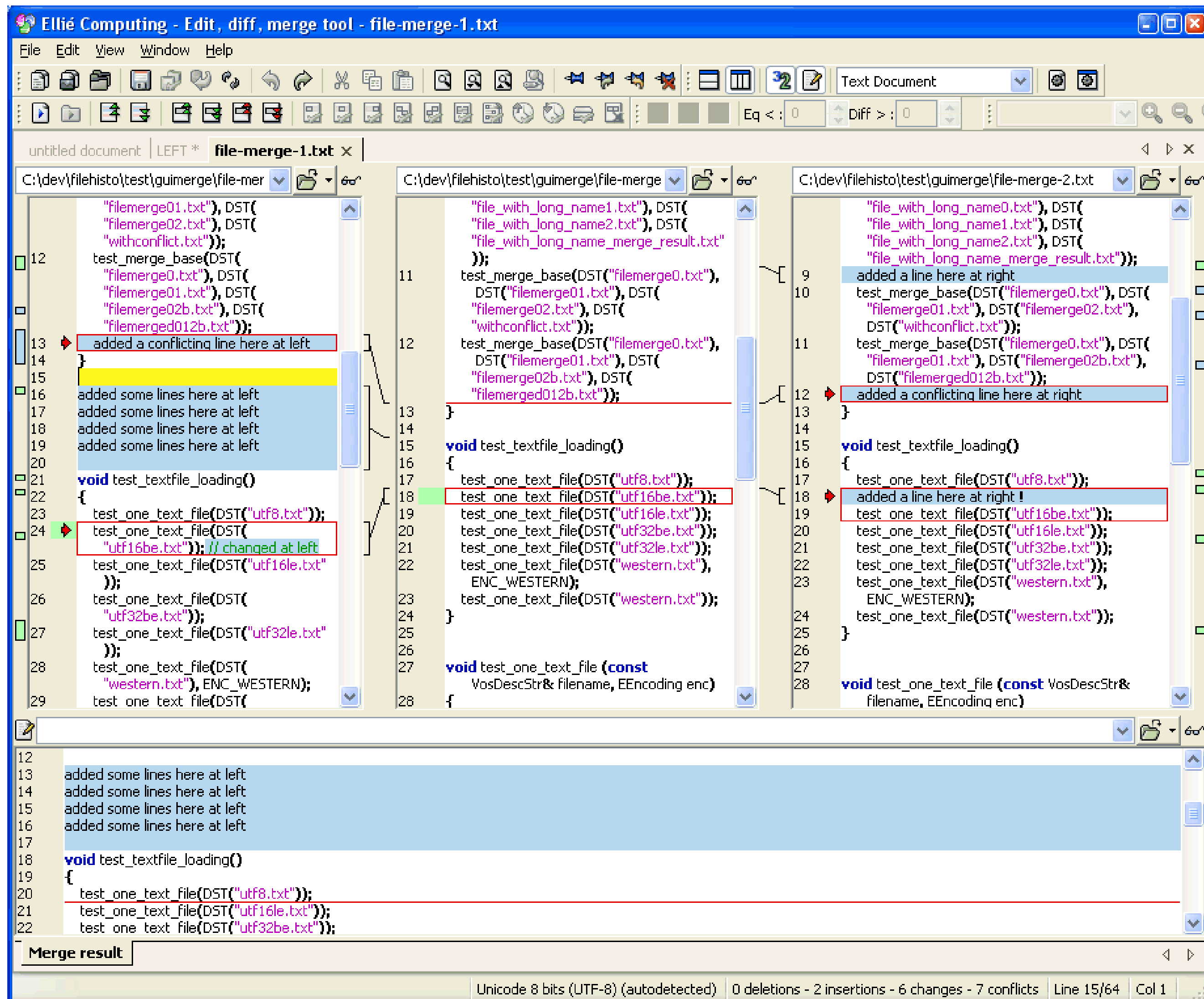
Workspace file: TagTemplate.pm

```
    } else {
        $tags = $self->{ TAGS };
    }

    # Loop until we have replaced all the tags.
        my $regex = $self->tag_pattern();
        while ( $string =~ /$regex/g ) {
                my $contents = $1;
        my $q_contents = quotemeta $contents;
        my $o_contents = $contents; # preserve in case we're ign
        # Remove leading and trailing whitespace.
        $contents =~ s/^\s+//;
        $contents =~ s/\s+$//;
        # Remove whitespace in quoted values.
        $contents =~ s|"([^"]*)"|
            my $value = $1;
            $value =~ s/ /\&#032;/g;
            $value =~ s/\t/\&#009;/g;
            $value =~ s/\n/\&#010;/g;
            $value =~ s/\r/\&#013;/g;
            $value =~ s/=/\&#061;/g;
            $value;
        |egm;
```

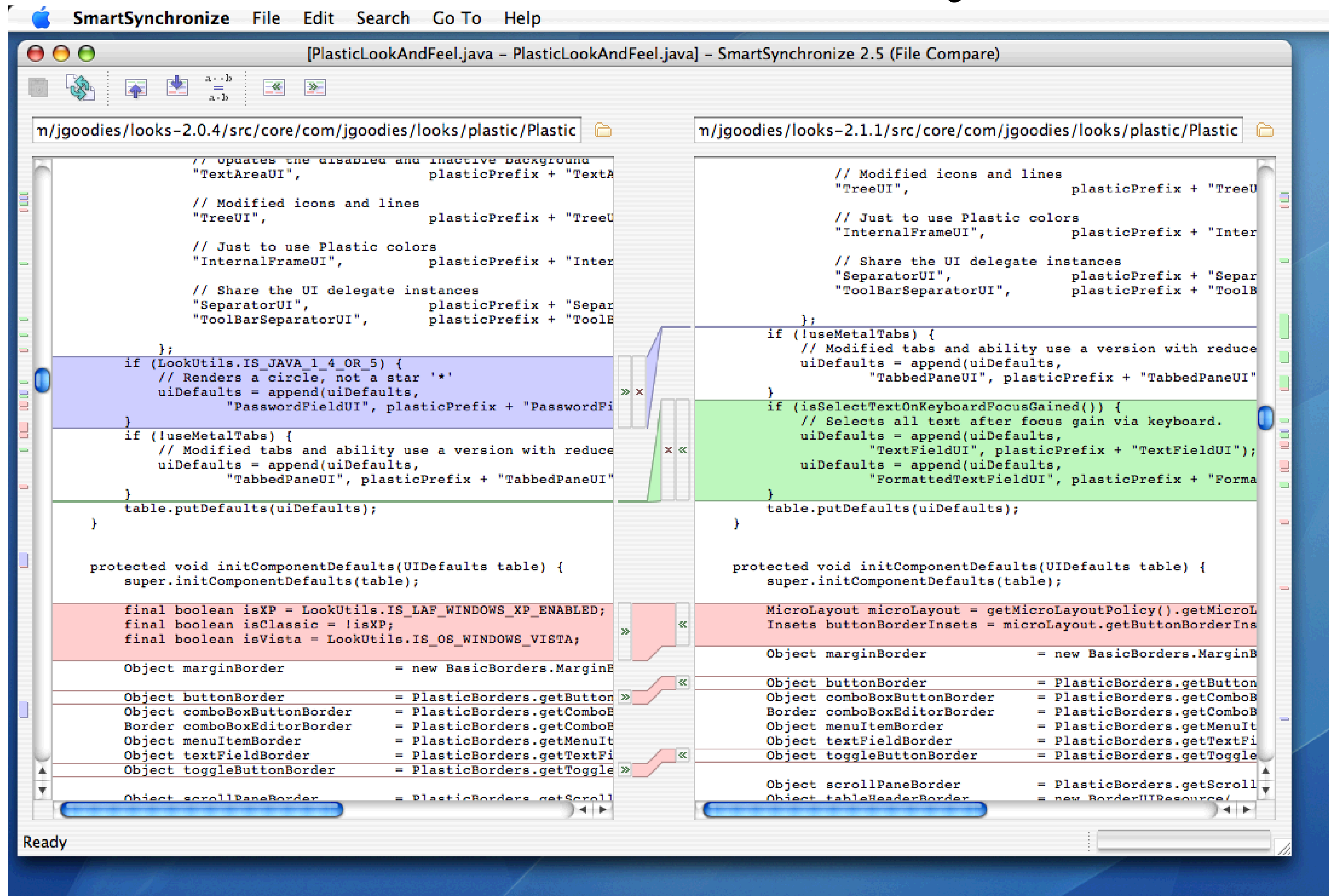
Repository file: TagTemplate.pm

```
    } else {
        $tags = $self->{ TAGS };
    }

    # Loop until we have replaced all the tags.
        while ( $string =~ /<#([^<>]*)>/g ) {
                my $contents = $1;
        my $q_contents = quotemeta $contents;
        my $o_contents = $contents; # preserve in case we're
        # Remove leading and trailing whitespace.
        $contents =~ s/^\s+//;
        $contents =~ s/\s+$//;
        # Remove whitespace in quoted values.
        $contents =~ s|"([^"]*)"|
            my $value = $1;
            $value =~ s/ /\&#032;/g;
            $value =~ s/\t/\&#009;/g;
            $value =~ s/\n/\&#010;/g;
            $value =~ s/\r/\&#013;/g;
            $value =~ s/=/\&#061;/g;
            $value;
        |egm;
        # Remove whitespace between parameters/equals-signs/
```

Other Version Control Systems?



Other Version Control Systems?



New Java features

Review of new Java features

- Java 5.0 introduced (at least) three language features
  - http://java.sun.com/j2se/1.5.0/docs/guide/language/
  - **Autoboxing/unboxing**
    - Automatically converts primitives (such as `int`) to wrapper classes (such as `Integer`)
  - **Enhanced `for` loop**
    - New concise syntax for iterating through a collection
  - **Generics**
    - Constrain what kinds of objects collections such as `Vector`s, `ArrayList`s, `Map`s, and `Set`s can contain
    - Removes need for annoying typecasts and `instanceof` checks

Before Autoboxing/unboxing

- Write a program that returns the sum of the elements in Integer array?

Autoboxing/unboxing

- This automatically creates an Integer object:
  - Integer i1 = 3; // autoboxing
  - int i2 = i1; // unboxing

- Note that this doesn't have a call to intValue:

```
/** Return the sum of the elements in a */
public static int sum(Integer[] a) {
    int result = 0;
    for (int i = 0; i != a.length; i++) {
        result += a[i];  // unboxing
    }
    return result;
}
```

Before Enhanced for loop

- Write a program that returns the sum of the elements in Integer array?!

Enhanced for loop

- Removes need for indexing:

```
– /** Return the sum of the elements in a */
– public static int sum(Integer[] a) {
–     int result = 0;
–     for (Integer i : a) {
–         result += i;
–     }
–     return result;
– }
```

- (Sure beats the old way, doesn't it?)

Enhanced for loop (cont.)

- Another example, using an ArrayList instead of an array:

```
/** Return the sum of the elements in a
  * Note the parameter type: more on following slides. */
public static int sum(ArrayList<Integer> a) {
    int result = 0;
    for (Integer v : a) {
        result += v;
    }
    return result;
}
```

- The old way:

```
Iterator iter = a.iterator();
while (iter.hasNext()) {
    Integer v = (Integer) iter.next();
    …
```

Before Java generics

- Write a program that iterates on a vector of strings and prints them ?

Java generics

- Java 1.4:
  - When you take something out of a collection (such as Vector) you must cast it
  - You can't say "I want a list of Dates"; instead, you must work with Objects
  - New feature: *generics*
- Java 1.5 generics:
  - Can specify what type a collection should contain *when you declare it*
  - Can only insert objects of that type
  - No need to cast when you get them back out

Java generics

- Examples of collections using generics:
  - java.util.Set<E>, java.util.List<E>
  - The <E> specifies a *generic type*
  - Can declare Sets and Lists of whatever we want
  - A list that can only hold Strings:
    - List<String> a = new ArrayList<String>();
    - a.add("Look ma, no cast on the next line");
    - String result = a.get(0);
    - a.add(new Date()); // compile error!

The sum() example again

- Our example uses generics, autoboxing, and the new for loop. Why is it good?

```
public static int sum(ArrayList<Integer> a) {
    int result = 0;
    for (Integer v : a) {
        result += v;
    }
    return result;
}
```

The sum() example again

- Our example uses generics, autoboxing, and the new for loop. Why is it good?

```
public static int sum(ArrayList<Integer> a) {
    int result = 0;
    for (Integer v : a) {
        result += v;
    }
    return result;
}
```

- New safety and ease:
  - Can only call sum with lists that contain only Integers
  - No need for a typecast or a call to intValue
  - No need to create an Iterator

A common problem

- Want to associate pairs of values where one of the values is guaranteed to be unique
- Example: match people with their favourite chocolate bar
  - "Michelle" => "Coffee Crisp"
  - "Paul" => "Kit Kat"
  - "Karen" => "Smarties"
- How would we do this with a list?

Solution: maps

- A list is a function from 0..N-1 to values
- As a list of pairs:
  - [["Darwin", "Snickers"], ["Newton", "Mars Bar"], ["Turing", "Kit Kat"]]
- A map is a function from keys to values
- As a map:
  - {"Newton"="Mars Bar", "Darwin"= "Snickers", "Turing"="Kit Kat"}
- Note: Maps are *unordered*

Maps

- Each key can appear at most once and has only one value
- Also called *hashes* (Perl), *dictionaries* (Python), and *associative arrays* (ancient)

Interface and Implementation

- Generic properties of maps defined by interface java.util.Map<K, V>: a Java interface that maps keys of type <K> to values of type <V>
- Classes implementing Map:
  - HashMap, TreeMap
  - Take CSC263H to learn how to choose

Putting values in

- To insert key/value pairs:
  - /**
  - * Associate value with key and return the
  - * previous value associated with key, or
  - * null if there was no previous mapping.
  - */
  - public V put(K key, V value)

- To get a printable version, use toString, which uses this format:
  - {key1=value1, key2=value2, ...}

Birthday example

```
public class Birthday {
    public static void main(String[] args) {
        Map<String, Integer> m =
            new HashMap<String, Integer>();
        m.put("Newton", 1642);
        m.put("Darwin", 1809);
        System.out.println(m);
    }
}
```

- {Darwin=1809, Newton=1642}

Getting values out

- To retrieve the value associated with a key:
  - /**
  - * Return the value associated with key,
  - * or null if the key is not in the map.
  - */
  - public V get(Object key)

- To check whether a key is in the map:
  - /**
  - * Return true if the map contains a value
  - * associated with key
  - */
  - public boolean containsKey(Object key)

Iterating

- Often iterate by getting the set of keys, and iterating over that

```
Set<String> keys = m.keySet();
Iterator<String> i = keys.iterator();
while (i.hasNext()) {
    String key = i.next();
    System.out.println
        (key + "=>" + m.get(key));
}
```

  - Darwin=>1809
  - Newton=>1642

Iterate using new for loop

- Or, using the new for loop:
  - Set<String> keys = m.keySet();
  - for (String key : keys) {
  -     System.out.println(
  -         key + "=>" + m.get(key));
  - }

Counting

```
public static void main(String[] args) {
    String[] data = "Be Mg Mg Ca Be Mg".split(" ");
    Map<String, Integer> m =
        new HashMap<String, Integer>();
    for (String d : data) {
      if (!m.containsKey(d)) {
        m.put(d, 1);
      } else {
        m.put(d, m.get(d) + 1);
      }
    }
    System.out.println(m);
}
```

{Ca=1, Mg=3, Be=2}

Inverting

```
public static void main(String[] args) {
    Map<String, String> byName =
        new HashMap<String, String>();
    byName.put("Darwin", "748-2797");
    byName.put("Newton", "748-9901");
    Map<String, String> byPhone =
        new HashMap<String, String>();
    for (Map.Entry<String, String> e : byName.entrySet()) {
        byPhone.put(e.getValue(), e.getKey());
    }
    System.out.println(byPhone);
}          {748-2797=Darwin, 748-9901=Newton}
```

Caution #1

- Do not modify key objects: location in map is computed from key contents (“hash code”)
  - Can't change Strings
  - But, you can change sets, lists, etc.
- What happens if you do?
  - Entry is now filed in the wrong location
  - May not be found the next time you search
  - Very hard to track down

Caution #2

- If you override method equals, override hashCode as well
  - a == b iff a and b are the same object
  - a.equals(b) checks to see if a and b refer to objects that have the same value
  - If a.equals(b), then a.hashCode() and b.hashCode() must return the same value
  - Why? Because that's how maps do lookups

Reminder: Standard input and output streams

- There are three file-like objects associated with every program:
  - stdin – standard input, usually from the keyboard（System.in）
  - stdout – standard output, usually to the screen（System.out）
  - stderr – standard error, usually to the screen（System.err）

Reminder: I/O

- BufferedReader input;
- input = new BufferedReader(new FileReader("filename"));
- input = new BufferedReader(new InputStreamReader(System.in));
- String line;
- while((line = input.readLine()) != null) {
- // do something
- }

Trick: reading from Strings

- We can use the same file abstraction to read from strings (and print to them):
  - input = new BufferedReader(
  - new StringReader("A string"));
- Primary advantage is for testing
- We can keep all test code in a single file, but use the same interfaces
- It is a lot faster

E1-Getting Started

- Work at school <u>first</u>
  - working at home is great, but
  - we mark your work on CDF
  - you need to learn Unix
- There are tools to help you work in multiple places (CVS, a version control system)

E1-Getting Started

- To work at home you need to install:
  - Microsoft Windows:
    - Cygwin – a Unix-like command interface for Microsoft Windows that provides CVS, ssh, editors, etc.
  - Every operating system:
    - Java – version 1.5
    - Eclipse – An IDE
      - Take the time now to learn it
    - JUnit
    - Python – version 2.4.1
  - Reminder: we mark everything on CDF
    - “It worked at home” won’t get you anywhere
- See “Useful links” on the course web page

E1-Getting Started

- Use Exercise 1 to learn new tools
  - Unix tools and programs
  - CVS
  - Eclipse
- Use the discussion board
  - ask for help installing software (ask your peers, and answer them)
- Get started on assignments early!

E1-Getting Started: Glass's Law

- *"Any new tool or working practice initially makes you less productive."*
- Always faster to solve the immediate problem the old way (in particular, DrJava)
- Don't let that stop you from learning new tools and skills (in particular, Eclipse)
- In the "real" world, you will have to re-train every three years