Programming Tools for Group Projects

Richard Smith
r.smith@cs.ucl.ac.uk

Lecture Plan

- Unit testing
- Version control
- Building
- Debugging
  - Logging
- Documenting
  - Code
  - Reports

General Biased advice

- Use a Unix based operating system
  - **Solaris** in the labs
  - **Mac OS X** if you are buying a laptop
  - Install **Linux** if you have a PC at home
- Learn how to do everything using the command line, *then* use an IDE!
  - Xemacs – difficult to learn, but integrates well with command line and has modes for everything
  - Eclipse – easier to learn, specialised for Java
- Use Java or C++ or 'script' (Python, Ruby, Perl) as appropriate

Unit testing

Background: Extreme Programming

- Pair programming
- Small releases
- Metaphor
- Simple design
- **Testing**
- Refactoring

www.extremeprogramming.org

Unit Testing

- Write tests before code.
- Code not done until tests all run.
- Re-run tests after any major changes
- Can be *confident* that nothing has broken.
- Hence not afraid to *refactor*.

Tools for Testing

- Some functional tests can be manual.
- Possible to automate most tests.
- Tools exist to make this easy and provide GUI.
- Java – **JUnit** – *www.junit.org*
- C++ – **CPPunit** – *cppunit.sf.net*

Installing CPPUnit

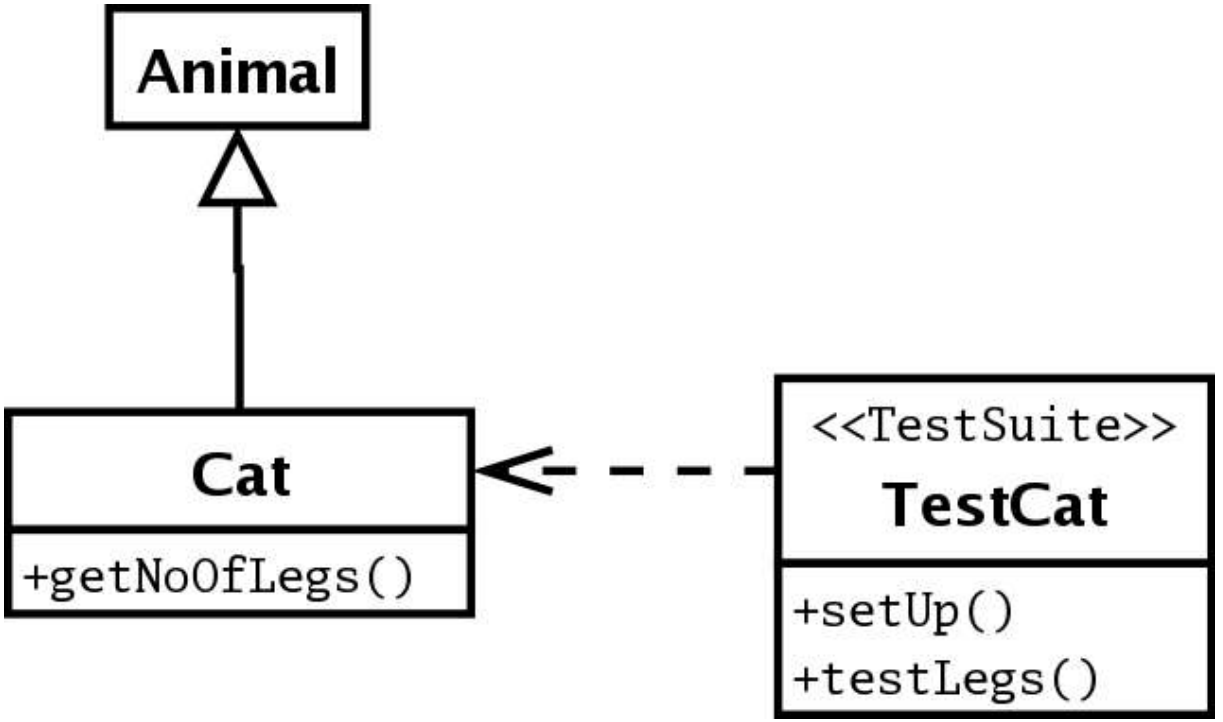
- We use GNU compiler on Unix-like OS (Linux, FreeBSD)
- `wget http://aleron.dl.sourceforge.net/sourceforge/cppunit/cppunit-1.10.2.tar.gz`
- `tar xvfz cppunit-1.10.2.tar.gz`
- `cd cppunit-1.10.2`
- `./configure; make`
- `su -c 'make install'`
- `su -c 'ldconfig'`
- Installation of GUI library is a bit more complicated :-(

Or you can use mine!

- bash
- export LD_LIBRARY_PATH=/cs/research/nets/home/marine/ucacrts/g2/local/lib
- export PATH=/cs/research/nets/home/marine/ucacrts/g2/local/bin:$PATH

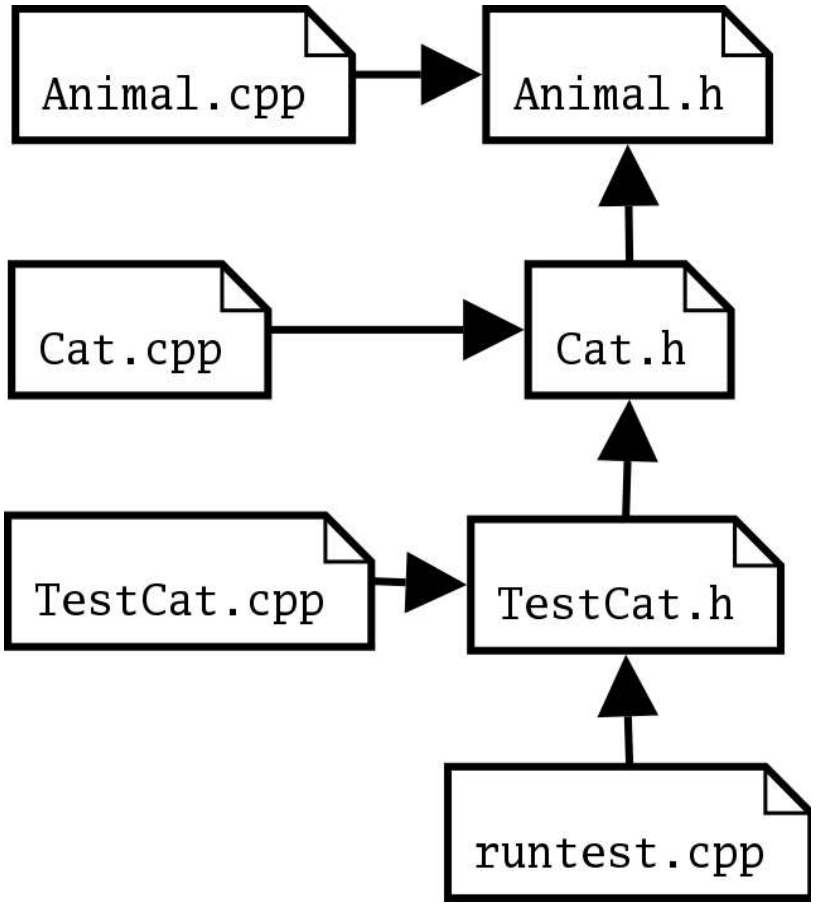
C++ Example

- Class diagram



C++ Example

- Source files (arrows show includes)



- Additional includes are not necessary, but are allowed, because of #IFNDEF guards

C++ Example

Animal.h

```
#ifndef ANIMAL_H
#define ANIMAL_H

class Animal{
 public:
  virtual int getNoOfLegs();
};
#endif
```

Cat.h

```
#ifndef CAT_H
#define CAT_H
#include "Animal.h"

class Cat: public Animal{
 public:
  virtual int getNoOfLegs();
};
#endif
```

Animal.cpp

```
#include "Animal.h"

int Animal::getNoOfLegs(){
  return 0;
}
```

Cat.cpp

```
#include "Cat.h"

int Cat::getNoOfLegs(){
  return 5;
}
```

C++ Example

TestCat.h

```
#ifndef TESTCAT_H
#define TESTCAT_H

#include <cppunit/extensions/HelperMacros.h>
#include "Cat.h"

class TestCat: public CPPUNIT_NS::TestFixture{
  CPPUNIT_TEST_SUITE(TestCat);
  CPPUNIT_TEST(testLegs);
  CPPUNIT_TEST_SUITE_END();
 public:
  void setUp();
 protected:
  void testLegs();
};

#endif
```


C++ Example

TestCat.cpp

```
#include "TestCat.h"

CPPUNIT_TEST_SUITE_REGISTRATION( TestCat );

void TestCat::testLegs(){
  Cat cat;
  CPPUNIT_ASSERT_EQUAL(4, cat.getNoOfLegs());
}

void TestCat::setUp(){
}
```

C++ Example

runtest.cpp

```
#include <cppunit/TextTestRunner.h>

#include "TestCat.h"

int main( int argc, char* argv[] ) {
  CppUnit::TextTestRunner runner;
  runner.addTest(TestCat::suite());
  bool result = runner.run("TestCat::testLegs", true, true, true);
  return result ? 0 : 1;
}
```

```
$> g++ -O2 -o runtest Animal.cpp Cat.cpp TestCat.cpp
runtest.cpp -ldl -lcppunit
```

CPPUnit Output Options

- *TestRunner* with *CompilerOutputter* – prints results suitable for use with IDE
- with *XMLOutputter* – results as XML document for processing by another application
- ***TextTestRunner*** – human readable output
- ***QtTestRunner*** – output in GUI
- *(MfcTestRunner, WxTestRunner)*

TextTestRunner Output

```
$> ./runtest
.F



!!!FAILURES!!!
Test Results:
Run:  1   Failures: 1   Errors: 0



1) test: TestCat::testLegs (F) line: 7
TestCat.cpp
equality assertion failed
- Expected: 4
- Actual  : 5
```


Success

```
$> ./runtest
.


OK (1 tests)


<RETURN> to continue
```


Installing CPPUnit QT

- Install tmake from *ftp.trolltech.com/freebies/tmake*
- After installing cppunit:
- `cd src/qttestrunner`
- `TMAKEPATH=/usr/lib/tmake/linux-g++ tmake qttestrunner.pro -o makefile`
- `make`
- `su -c 'cp -d ../../lib/* /usr/local/lib'`
- `su -c 'ldconfig'`

QT GUI Example

runtestgui.cpp

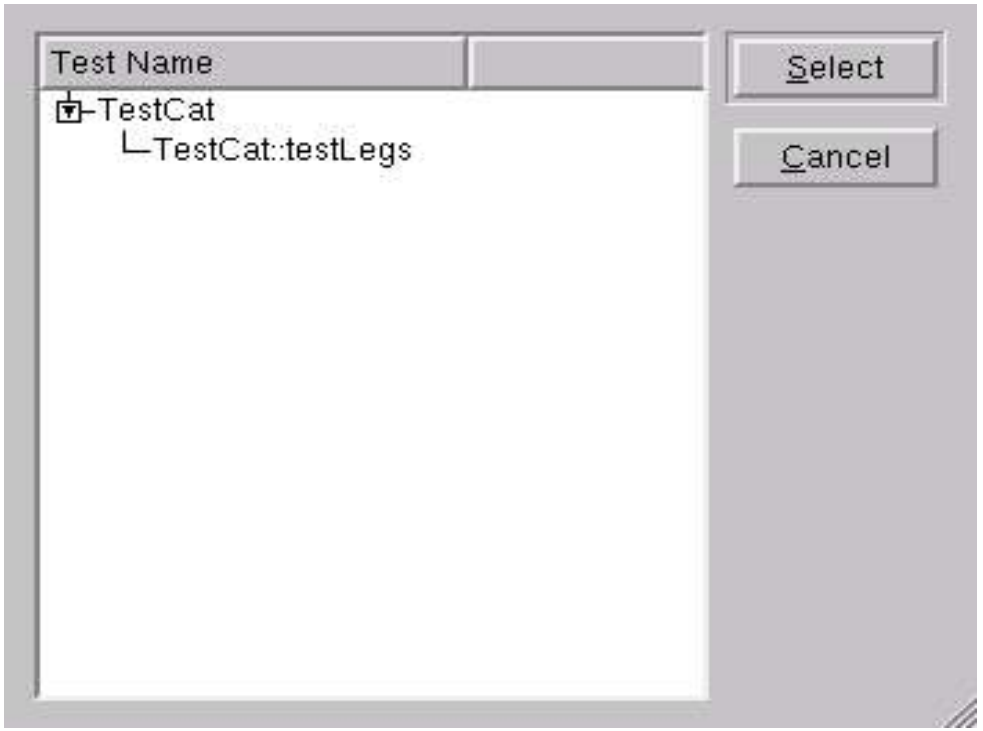
```
#include <qapplication.h>
#define QTTESTRUNNER_API __declspec(dllimport)
#include <cppunit/ui/qt/TestRunner.h>

#include "TestCat.h"

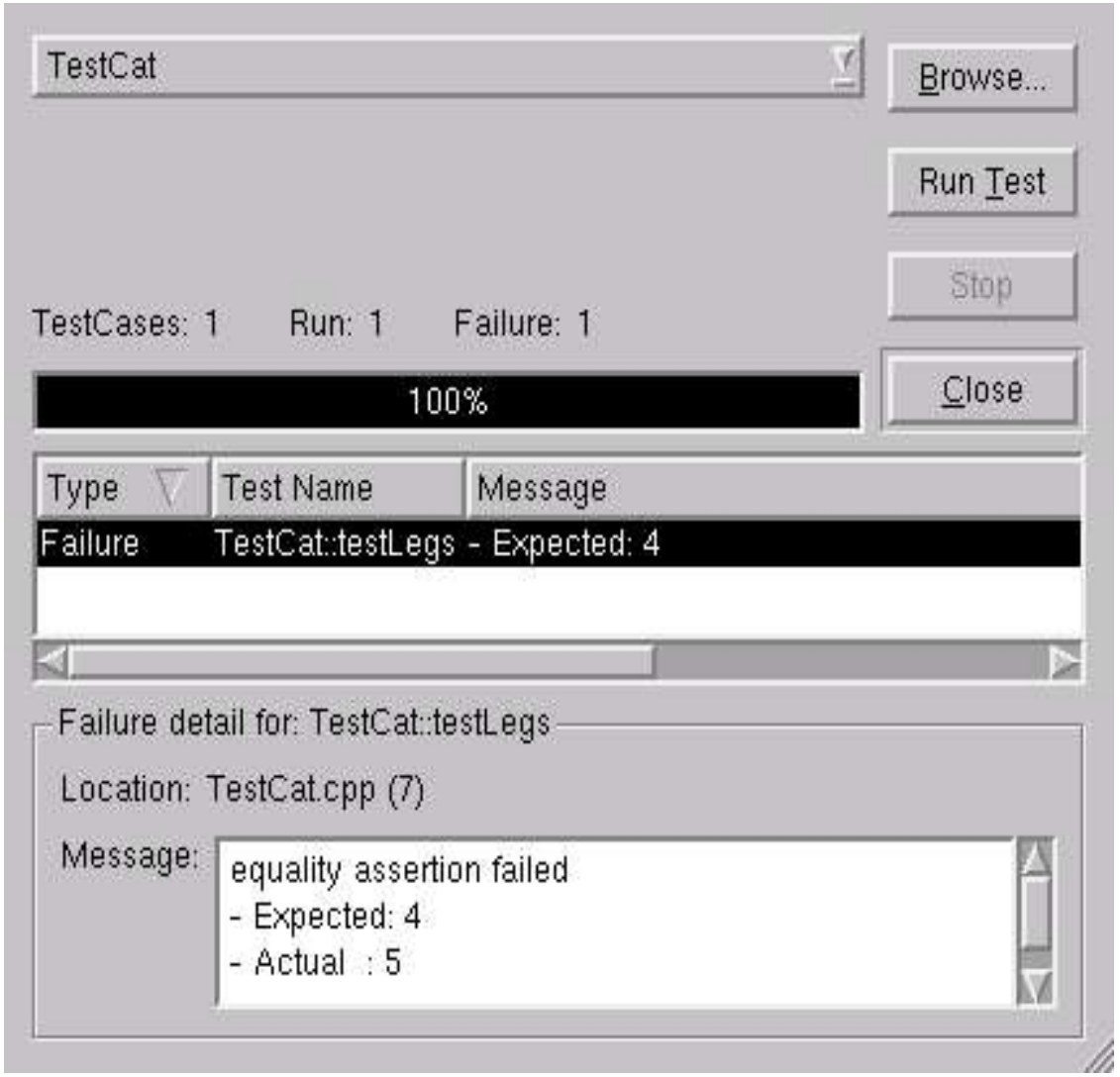
int main( int argc, char* argv[] ) {
  QApplication app( argc, argv );
  CppUnit::QtTestRunner runner;
  runner.addTest( TestCat::suite() );
  runner.run(true);
  return 0;
}
```


QT GUI

```
$> g++ -O2 -o runtestgui Animal.cpp Cat.cpp
TestCat.cpp runtestgui.cpp -L/usr/qt/3/lib -I/
usr/qt/3/include -lqt-mt -ldl -lcppunit
-lqttestrunner
```

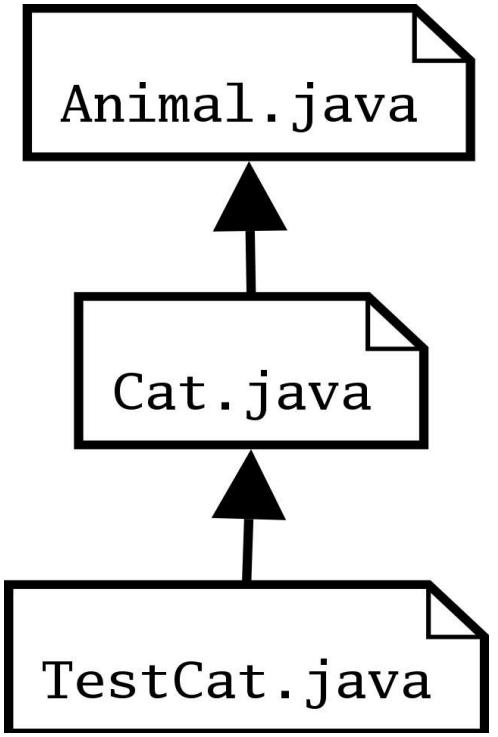


QT GUI Results



Java Example

- Source files (no #includes!)



Java Example

Animal.java

```
class Animal{
    public int getNoOfLegs(){
        return 0;
    }
}
```

Cat.java

```
class Cat extends Animal{
    public int getNoOfLegs(){
        return 5;
    }
}
```

Java Example

TestCat.java

```
import junit.framework.*;

public class TestCat extends TestCase{

    public void testLegs(){
        Cat cat = new Cat();
        Assert.assertEquals(4, cat.getNoOfLegs());
    }

    protected void setUp(){}
}
```


JUnit Output Options

- We don't need to write a tester program – provided program will load our class at run-time.
- *junit.textui.TestRunner* – text output
- *junit.awtui.TestRunner* – GUI output
- *junit.swingui.TestRunner* – Nicer GUI output
- Further options, such as XML output, provided by Ant.

JUnit Example

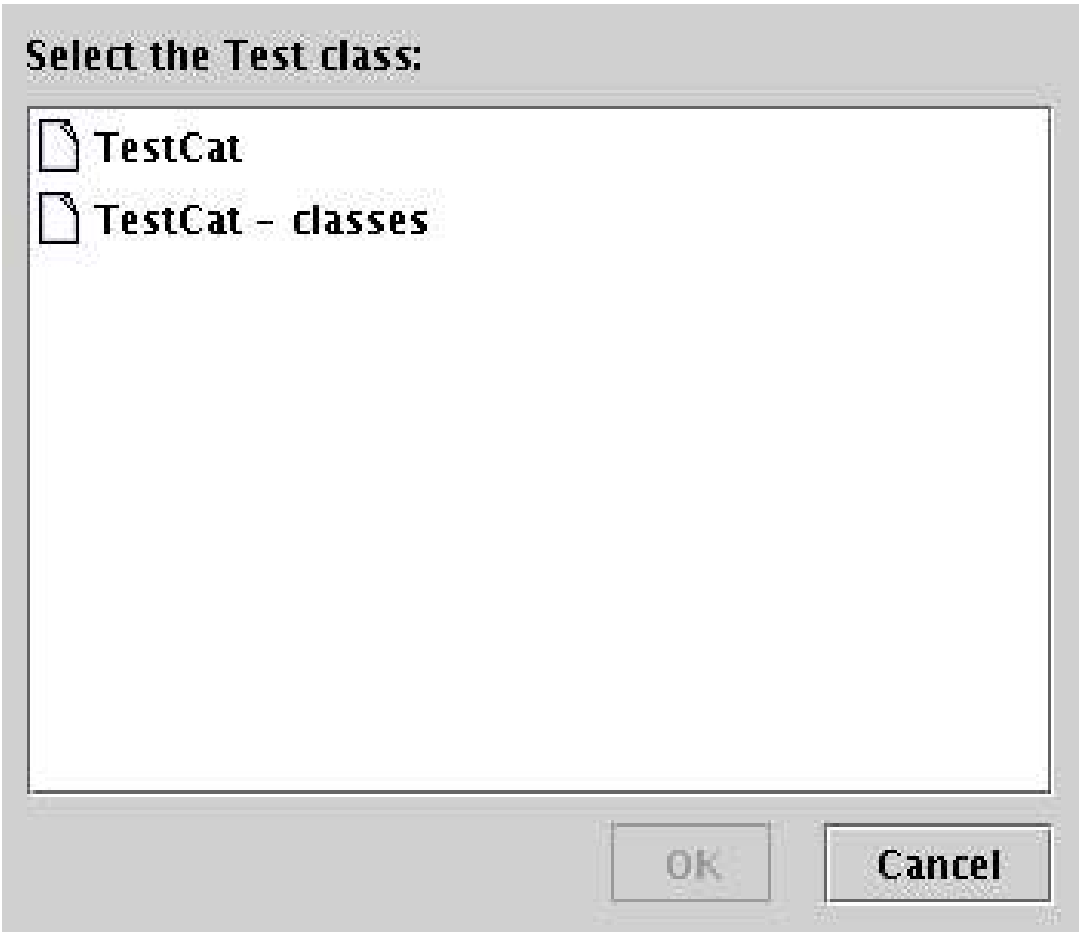
- `javac TestCat.java`
- Invoking JUnit is much easier than CPPUnit.
- TestRunner programs are provided which load your bytecode via reflection, no need to compile your own runners.
- `java junit.textui.TestRunner TestCat`
- `java junit.awtui.TestRunner TestCat`
- `java junit.swingui.TestRunner TestCat`

JUnit Results

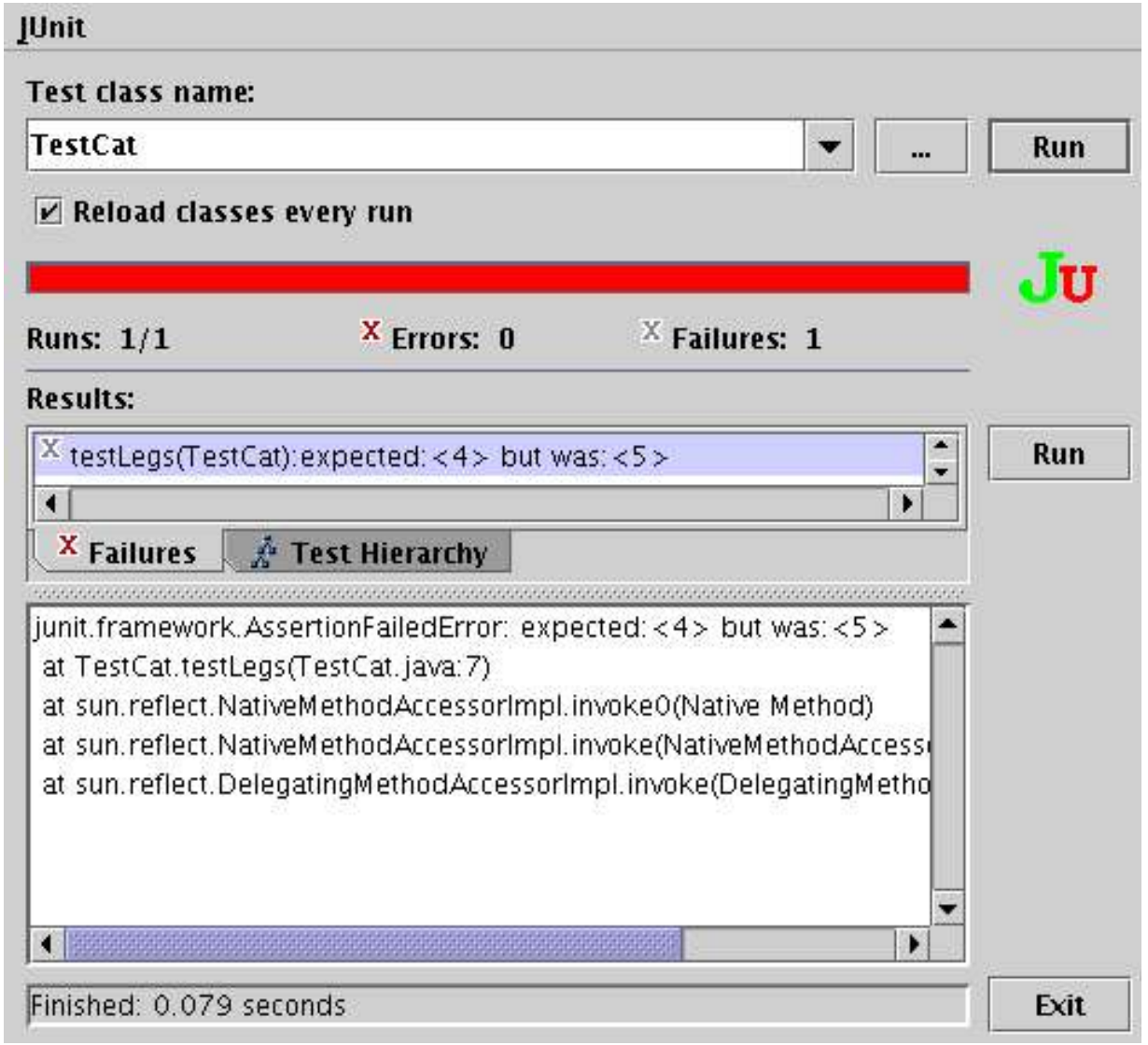
```
$> java junit.textui.TestRunner TestCat
.F
Time: 0.006
There was 1 failure:
1) testLegs(TestCat)junit.framework.AssertionFailedError:
expected:<4> but was:<5>
        at TestCat.testLegs(TestCat.java:7)
        at sun.reflect.NativeMethodAccessorImpl.invoke0
(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke
(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke
(DelegatingMethodAccessorImpl.java:25)

FAILURES!!!
Tests run: 1,  Failures: 1,  Errors: 0
```


JUnit GUI



JUnit GUI Results



Questions?

Building

Building

- Large projects consist of many source files which must be compiled in the correct order.
- This can take hours. If only a few files have been edited, we don't need to recompile the whole lot.
- But how do we know which ones to recompile?

Make

- GNU Make is the most popular tool to solve this problem.
- You write 'Makefile' which describes all the dependencies between the source files.
- You run Make which then runs compiler for you.
- Only files that have changed (and their dependants) are recompiled.

Make: additional benefits

- Makefile stores info (e.g. flags) making it easy for others to compile your program.
- If you want to change all flags, e.g. to add debug info, only one change needed.
- Make can invoke any command, not just compiler.
- Common uses: installation, removing object files, generating docs, packaging releases

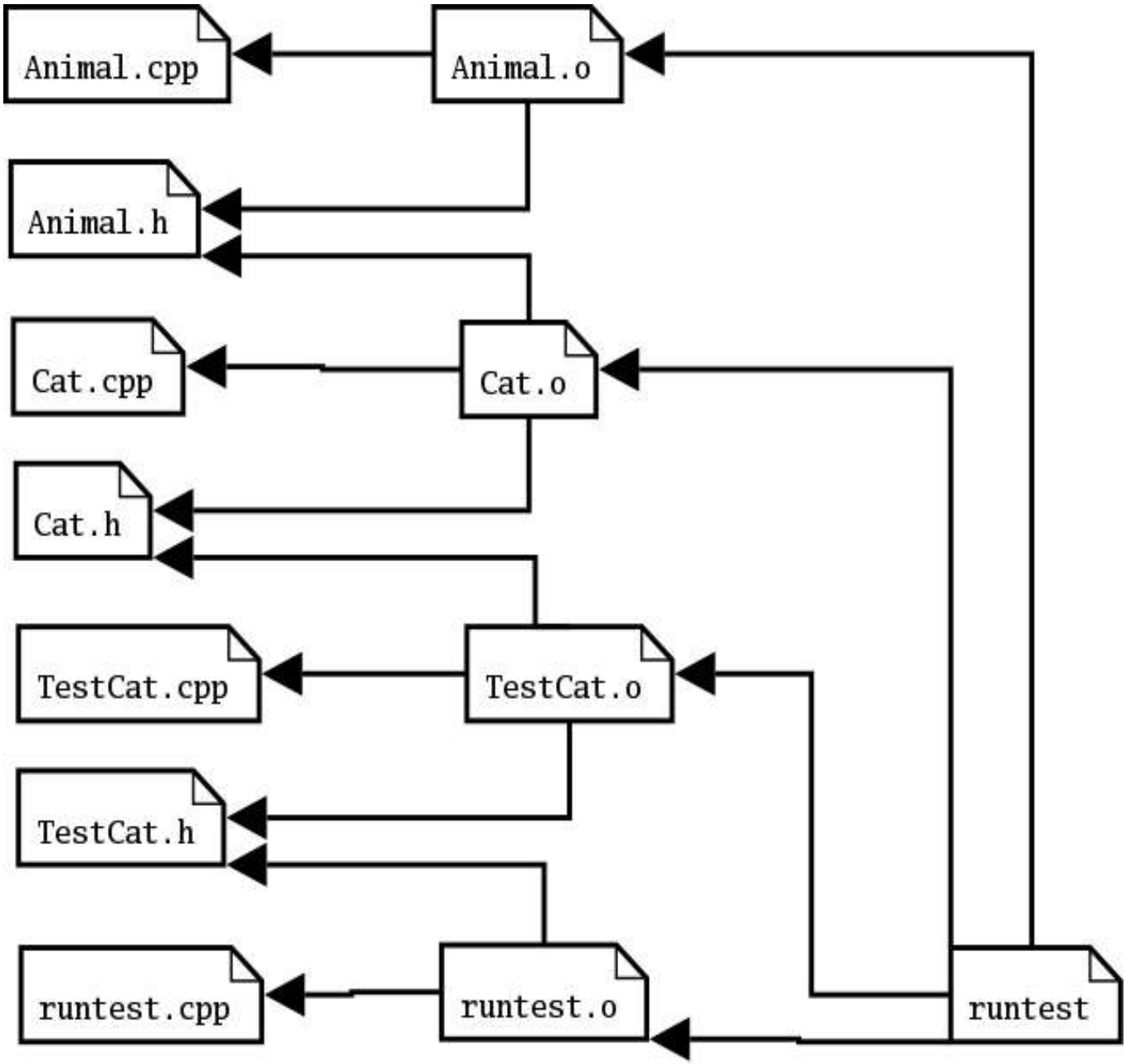
Makefile rules

- <target>: <dependencies>
  <action>
- Note the second line must be indented with a **tab** (not spaces)
- *target* – name of object file (usually)
- *dependencies* – list of other targets
- *action* – command to update target

Make Behaviour

- A target is up-to-date if
  - a file of the name exists
  - none of the files listed as dependencies were modified since it was last updated
  - all its dependencies are up-to-date
- If it is not up-to-date, first its dependencies are updated (recursively), then it is updated, by executing *action*.
- Make begins by bringing up-to-date the target named on the command line. If none is supplied, Make uses the first target in the Makefile.

C++ Dependency Tree



Makefile

```
runtest: Animal.o Cat.o TestCat.o runtest.o
	g++ runtest.o TestCat.o Cat.o Animal.o -o runtest -ldl -lcppunit
runtestgui: Animal.o Cat.o TestCat.o runtestgui.o
	g++ runtestgui.cpp TestCat.o Cat.o Animal.o -o runtestgui -ldl \
	-lcppunit  -L/usr/qt/3/lib -lqt-mt -lqttestrunner

Animal.o: Animal.cpp Animal.h
	g++ Animal.cpp -c -o Animal.o
Cat.o: Cat.cpp Cat.h Animal.h
	g++ Cat.cpp -c -o Cat.o
TestCat.o: TestCat.cpp TestCat.h Cat.h
	g++ TestCat.cpp -c -o TestCat.o
runtest.o: TestCat.h runtest.cpp
	g++ runtest.cpp -c -o runtest.o
runtestgui.o: TestCat.h
	g++ runtestgui.cpp -c -o runtestgui.o -I/usr/qt/3/include
```

Implicit rules

- Make is (slightly) intelligent – it can infer obvious actions.
- `blah.o: blah.c`
  implies
  `blah.o: blah.c`
  `gcc -c -o blah.o blah.c`
- If you do not specify any rule for a dependency, Make will search its list of implicit rules and attempt to find one that will produce the needed file.
- Thus you can use Make with no Makefile at all!
  `$> make test`
  `gcc -o test test.c`

Environment Variables

- Inherited from shell environment
- Define: MY_VAR = some text
- Use: $(MY_VAR)
- Implicit rules make use of many variables, e.g.
  - CFLAGS – options for C compiler
  - LDFLAGS – options for linker

Pseudo-targets

- **Pseudo-targets** are not files, and hence can never be up-to-date. Specifying a pseudo-target on the command line will always force make to update the listed dependencies and execute the action.
- Common pseudo-targets
  - *default* (must be first target) to specify which components are built when no arguments are given
  - *clean* to delete all object files

Improved Makefile

```
OBJECTS = Animal.o Cat.o TestCat.o
LDFLAGS = -ldl -lcppunit -L/usr/qt/3/lib -lqt-mt -lqttestrunner
CPPFLAGS = -I/usr/qt/3/include

default: runtest runtestgui

runtest: $(OBJECTS) runtest.o
runtestgui: $(OBJECTS) runtestgui.o

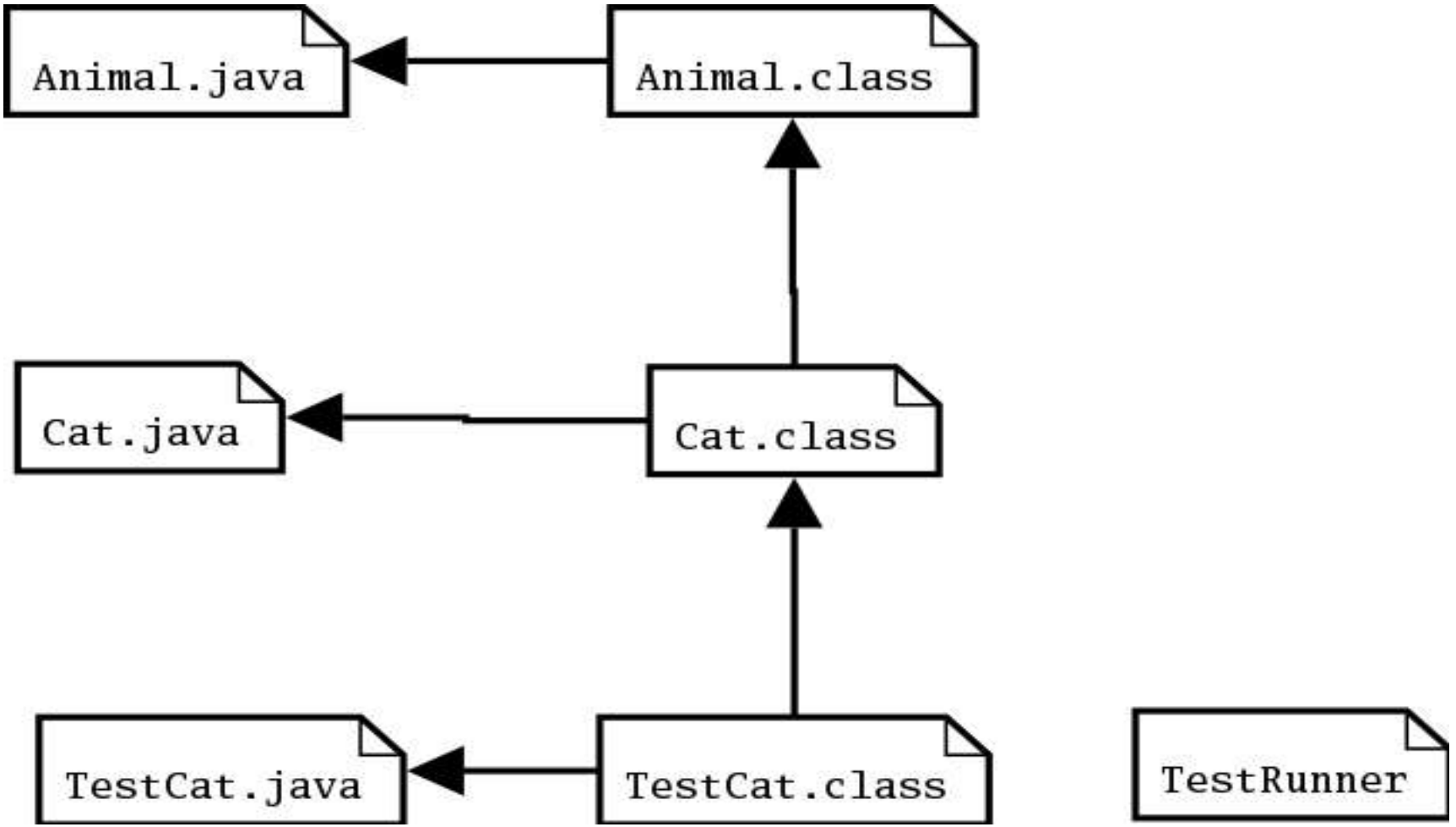
Animal.o: Animal.cpp Animal.h
Cat.o: Cat.cpp Cat.h Animal.h
TestCat.o: TestCat.cpp TestCat.h Cat.h
runtest.o: TestCat.h runtest.cpp
runtestgui.o: TestCat.h runtest.cpp

clean:
	-rm -f $(OBJECTS) runtest.o runtest runtestgui.o runtestgui
.PHONY: clean
```

Other uses for Make

- Packaging distribution
- Running program
- Running tests automatically
- Version control operations
- Running Make again in sub-directories
- Saving you from re-typing long commands

Java Dependency Tree



Makefile generic rules

- Make knows how to compile many languages by default, but it is easy to add new ones.
-
```
%.class : %.java
        javac $<
```
-
```
%.pdf : %.ps
        ps2pdf $< $@
```


Java Makefile

```
%.class : %.java
	javac $< -d /my/classes -sourcepath /my/sources

CLASSPATH=/usr/share/junit/lib/junit.jar:.

default: testtext

testtext: TestCat.class
	java junit.textui.TestRunner TestCat
testgui: TestCat.class
	java junit.swingui.TestRunner TestCat

TestCat.class: TestCat.java Cat.class
Animal.class: Animal.java
Cat.class: Cat.java Animal.class

clean:
	-rm -f *.class
.PHONY: clean
```


Make & Java

- Make doesn't support Java by default
- Make methodology of invoking compiler for each object does not suit Java – long VM startup time
- Java compiler has its own partial dependency checking – confuses Make
- Lack of header files
  - C++ dependants only recompiled when API changes. Java does not know whether API or implementation changed.
  - More difficult to generate your dependency tree without #includes

Make - shortcomings

- Many different ways to write a Makefile – wastes time
- Must remember to keep dependencies updated – else strange bugs appear
- Circular dependencies not allowed
- Large projects require multiple Makefiles with hardcoded paths – not very portable
- Solutions
  - Hack on top of Make: Automake/Autoconf
  - Use more modern alternative

Alternatives

- Automake and Autoconf - *www.gnu.org*
- Cmake – Cross Platform Make – *www.cmake.org*
- JMK – Make in Java – *jmk.sf.net*
- Apache Ant - *ant.apach.org*
- many more

Ant

- Higher level – deals with tasks rather than individual files
- Automatically generates dependencies
  (This is possible for C++ using Automake and Autoconf, but Ant is easier to learn.)
- Compiles all files using single VM so much faster than Make for Java projects.
- Extended via Java classes, not shell commands

Ant build.xml example

```
<?xml version="1.0" ?>
<project name="example" default="compile">
 <description>Example</description>

 <path id="compile.classpath">
     <pathelement location="/usr/share/junit/lib/junit.jar"/>
 </path>

 <target name="compile" description="Compiles source code">
  <depend srcdir="." cache="depend.cache" closure="true">
      <classpath refid="compile.classpath" />
  </depend>
  <javac srcdir=".">
      <classpath refid="compile.classpath" />
  </javac>
 </target>

</project>
```


Running Ant

- Run with build.xml
  - `ant`
- Run with different build file
  - `ant -f mybuild.xml`
- Run with build file in some parent directory
  - `ant -find build.xml`

Buildfile basics

- XML file containing list of *targets*
- Each target represents one or more high level *tasks*, e.g. compile, run
- Specify description and default target
  - <?xml version="1.0" ?>
    <project name="myProj" default="compile">
    <description>Compiles our project </description>
- View description and tasks: `ant -p`
- Override default target: `ant` *`myTarget`*

Ant paths

- Can include jar archives and directories, just like command-line classpath does:

```
<path id="myVeryOwnClassPath">
    <pathelement location="classes"/>
    <pathelement location="libs/myLib.jar">
</path>
```

- But may also include previously defined paths

```
<path id="myRunTimeClassPath">
    <path refidid="myVeryOwnClassPath">
    <pathelement location="libs/runtime.jar">
</path>
```

Ant Paths (2)

Also wildcards

```
<path id=”anotherPath”>
    <fileset dir=”mylibs”>
        <include name=”*.jar”>
        <exclude name=”notThisOne.jar”>
    </fileset>
</path>
```


Compiling with Ant: *depend* task

- Compile target usually contains two tasks.
- First we delete all the class files that have dependencies which are out of date

```
<target name="compile" description="Compiles  code">
   <depend srcdir="src" destdir="classes"
     cache="depend.cache" closure="true">
        <classpath refid="myVeryOwnClassPath" />
   </depend>
```

Compiling: *javac* task

- Then we compile to
  - Update out-of-date class files
  - Create missing class files

```
    <javac srcdir="src" destdir="classes" debug="true">
      <classpath refid="myVeryOwnClassPath" />
    </javac>
</target>
```

Running with Ant: Java task

- Targets can depend on other tasks, e.g. The *run* target cannot be performed until after the *compile* target.

```
<target name="run" depends="compile">
  <java classname="myMainClass" fork="true">
    <classpath refid="myRuntimeClasspath" />
  </java>
</target>
```

Other tasks

- Mkdir
- Delete
- Splash – displays image and progress bar
- Junit – integrates unit testing
- See following example for details
- Remember you can code custom tasks in Java

Improved build.xml (1)

```
<?xml version="1.0" ?>
<project name="example"
default="compile">
  <description>Example</description>

  <path id="compile.classpath">
      <pathelement
location="/usr/share/junit/lib/junit.jar"/>
  </path>

  <path id="run.classpath">
    <path refid="compile.classpath" />
    <pathelement location="classes/" />
  </path>

  <target name="init">
    <mkdir dir="classes" />
  </target>
```


Improved build.xml (2)

```
<target name="compile" depends="init" description="Compiles
source code">
    <splash
imageurl="http://www.cs.ucl.ac.uk/orange_roll/orange_banner.gif"
 />
    <depend srcdir="." destdir="classes" cache="depend.cache"
  closure="true">
        <classpath refid="compile.classpath" />
    </depend>
    <javac srcdir="." destdir="classes">
        <classpath refid="compile.classpath" />
    </javac>
  </target>
```


Improved build.xml (3)

```
<target name="test" depends="compile">
   <junit printsummary="true">
     <classpath refid="run.classpath" />
     <test name="TestCat" />
     <formatter type="brief" usefile="false" />
   </junit>
 </target>

 <target name="clean" depends="init">
   <delete dir="classes" />
 </target>

</project>
```

Properties

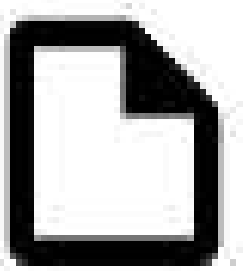
- Properties are parameters specified on command line
- Junit task example

```
<test name="${testcase}" if="testcase"/>
<test name="TestAll" unless="testcase"/>
```

- To run all tests:
  - ant
- To run particular test:
  - ant -Dtestcase=*thisTest*

Questions?

Version Control



Working on same file

Fred

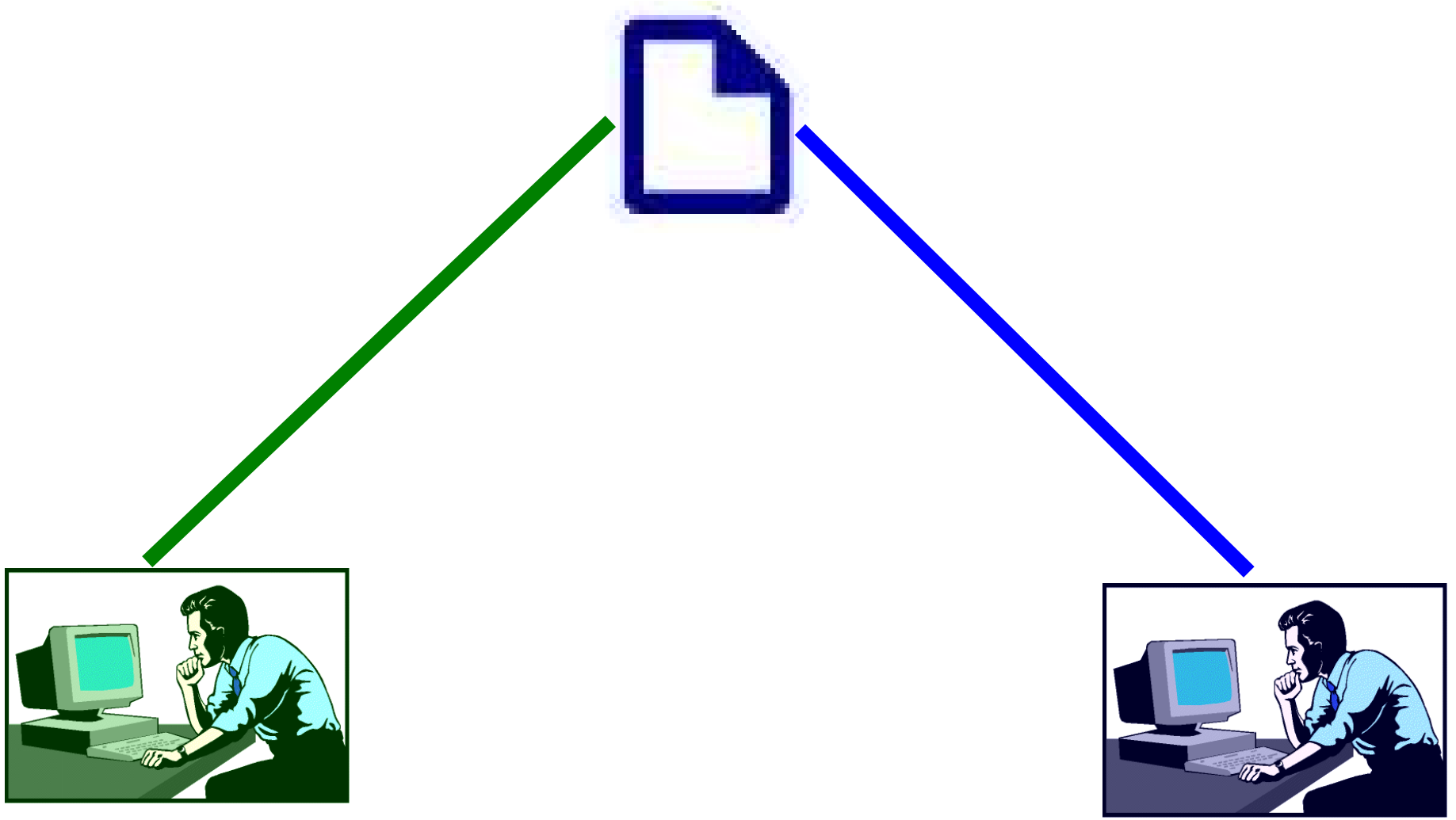
John

Version Control

- Two programmers, Fred and John, working on one program.
- What if both want to edit same file at same time? Do we lock the file? (Then Fred has to wait for John to finish.)
- What if Fred breaks the program? Restore from yesterday's backup? (And lose all work done by John today?)

Fred

John



Fred

John

John has overwritten my file!

Fred

John

Fred has locked me out!

CVS

- Built on top of older single-file based system RCS, a bit clunky
- But free and already installed on every Unix machine
- Ubiquitous standard for distributed Open Source development
- Free book at *cvsbook.red-bean.com*
- Many tools, GUIs (Cervisia), and ports
  (for Microsoft see WinCVS and TortoiseCVS)

CVS Example

- Configure for repository on local filesystem:
  `export CVSROOT=/home/richard/cvs`
- Or remote system:
  ```
  export
  CVSROOT=:ext:user@example.com:/home/user/cvs
  export CVS_RSH=ssh
  ```
- Initialise repository (only once!):
  cvs init

Import

Initial import:

```
cd ~/animals
cvs import -m “initial import” animals vendor start
```

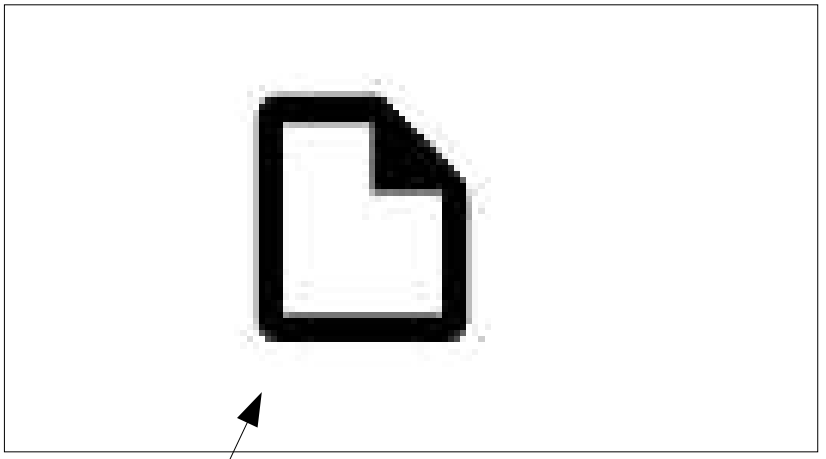
```
N animals/Animal.cpp
N animals/Animal.h
N animals/Makefile
N animals/Cat.cpp
N animals/Cat.h
N animals/TestCat.cpp
N animals/TestCat.h
N animals/runtestgui.cpp
N animals/runtest.cpp
N animals/Makefile2

No conflicts created by this import
```

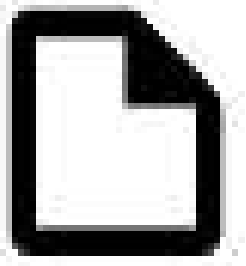

Import

- `cvs import -m “`*`log msg`*`” `*`projectname vendortag releasetag`*
- If *log msg* is omitted, editor will load for you to enter one.
- *projectname* usually matches directory name
- *vendortag* and *releasetag* do not matter*

CVS
respository



import



Fred

John

Checkout

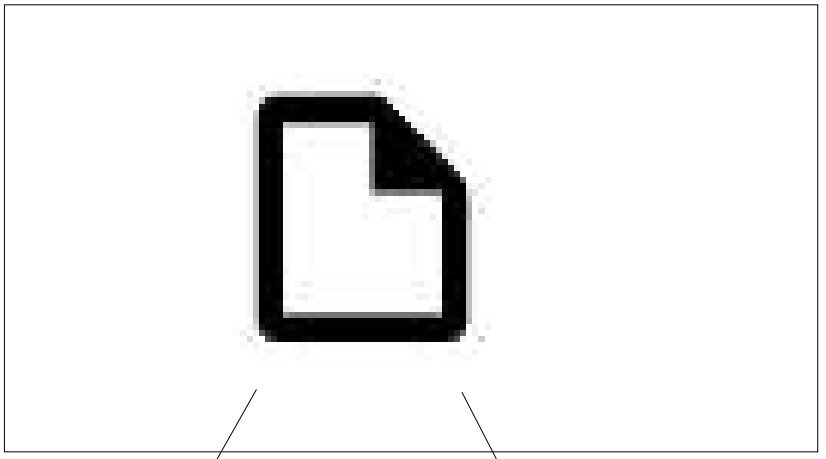
- We must checkout a copy of the project before we begin work (~/animals is not modified by import)
- ```
  cd
  mv animals animals_backup
  cvs checkout animals
  ```
- 'U' flag shows files updated.
- Good practice to release after work finished:
  ```
  cvs release -d animals
  ```
  ```

Checkout

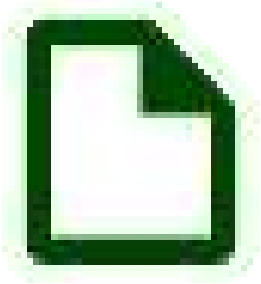
```
cvs checkout: Updating animals

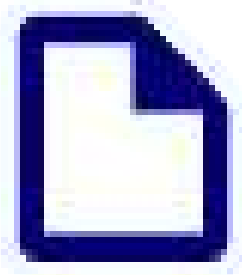
U animals/Animal.cpp
U animals/Animal.h
U animals/Cat.cpp
U animals/Cat.h
U animals/Makefile
U animals/Makefile2
U animals/TestCat.cpp
U animals/TestCat.h
U animals/runtest.cpp
U animals/runtestgui.cpp
```


CVS
respository



checkout





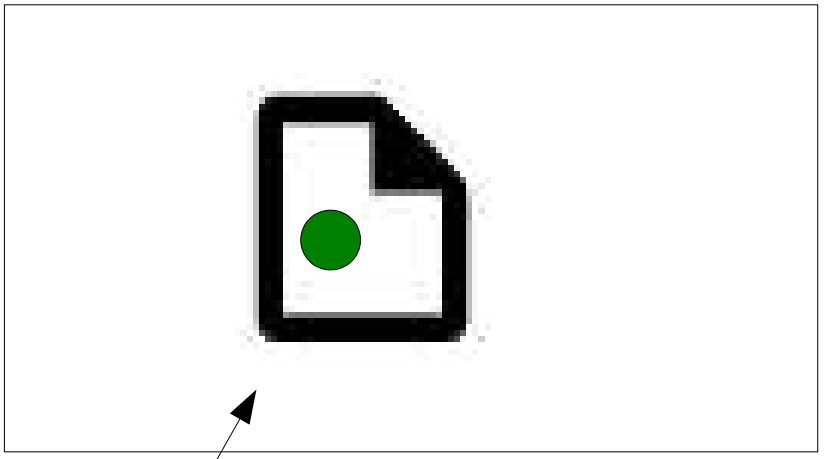
Fred

John

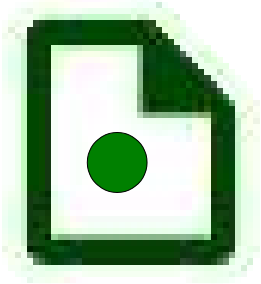
Committing changes

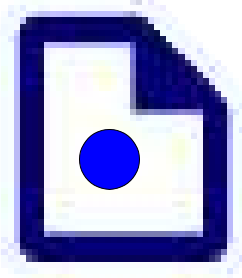
- Now you can edit the files.
- When you have successfully implemented a feature:
  `cvs commit -m “blah is working”`
- If you add or remove files:
  ```
  cvs add file.c
  cvs remove file.c
  cvs commit -m “added/removed blah”
  ```


CVS
respository



commit

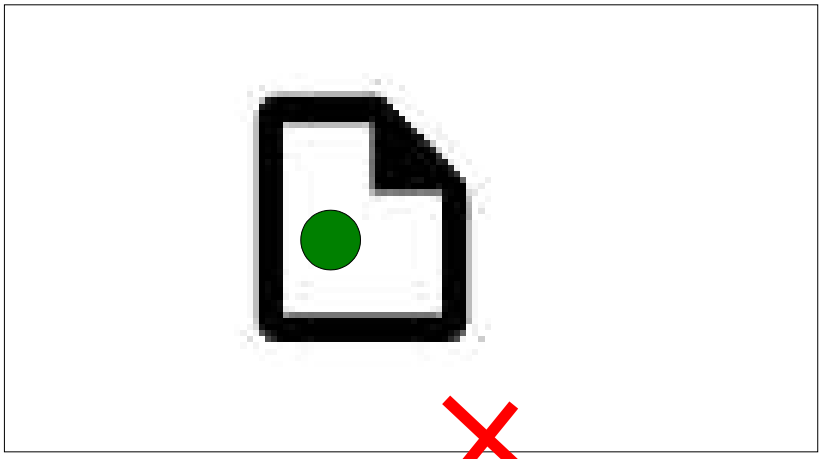




Fred

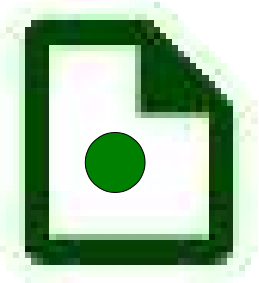
John

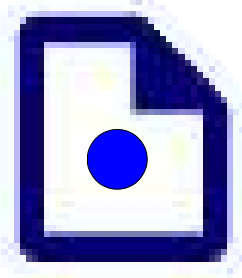
CVS
respository



commit

Commit fails
because local copy
not up to date

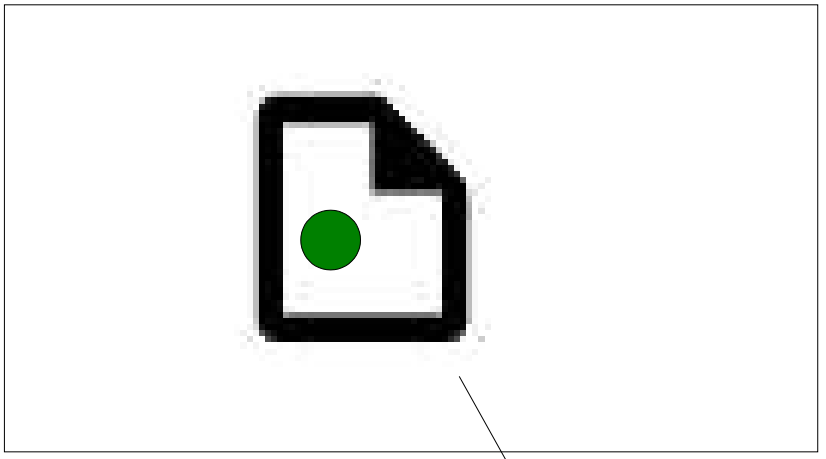




Fred

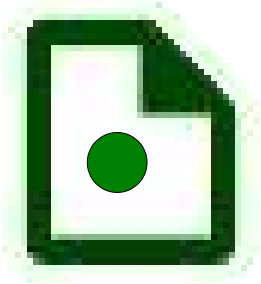
John

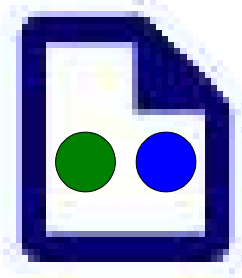
CVS
respository



update

changes merged!

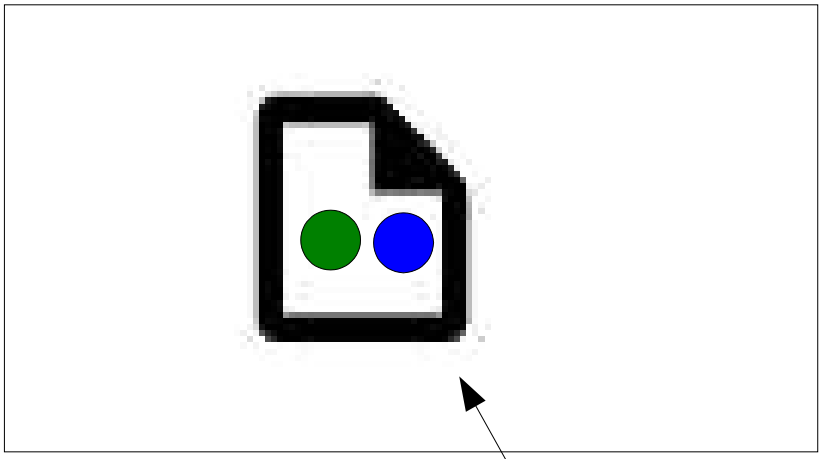




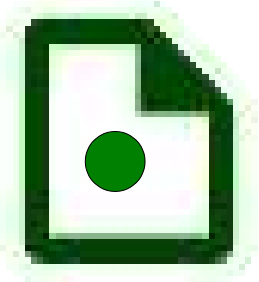
Fred

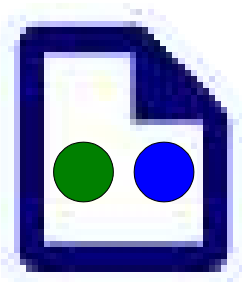
John

CVS
respository



commit





Fred

John

Fred & John

- Fred edits Cat.cpp to fix the number of legs (i.e. he changes '5' to '4'), then commits.

```
Checking in Cat.cpp;
/home/richard/cvs/animals/Cat.cpp,v  <--  Cat.cpp
new revision: 1.2; previous revision: 1.1
done
```

Fred & John

- Meanwhile, John is editing the same file, and adds a descriptive comment at the top:
  / Felis domesticus */*
- John attempts to commit his changes, but:

```
cvs commit: Examining .
cvs commit: Up-to-date check failed for `Cat.cpp'
cvs [commit aborted]: correct above errors first!
```

Fred & John

- John must bring his Cat.cpp up to date so it includes Fred's changes:
  `cvs update`

```
cvs update: Updating .
RCS file: /home/richard/cvs/animals/Cat.cpp,v
retrieving revision 1.1
retrieving revision 1.2
Merging differences between 1.1 and 1.2 into Cat.cpp
M Cat.cpp
```


Fred & John

- CVS has combined both changes in John's working copy:

```
/* Felis domesticus */
#include "Cat.h"

int Cat::getNoOfLegs()
{
  return 4;
}
```

- John may now commit the changes to the repository.

Conflict

- What if Fred and John both edit the same line of the same file? e.g. Fred decides that the Cat will have 8 legs, while John changes it to 10?

```
cvs update: Updating .
RCS file: /home/richard/cvs/animals/Cat.cpp,v
retrieving revision 1.3
retrieving revision 1.4
Merging differences between 1.3 and 1.4 into Cat.cpp
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in Cat.cpp
C Cat.cpp
```

Resolving Conflict

- CVS requires you to fix it by hand before you can commit again:

```
/* Felis domesticus */
#include "Cat.h"

int Cat::getNoOfLegs(){
<<<<<<< Cat.cpp
  return 8;
=======
  return 10;
>>>>>>> 1.5
}
```

More CVS commands

- cvs diff -r 1.1 -r 1.2 file
- cvs status file
- cvs history
- cvs log file

How it works

- Metadata is stored in 'CVS' directories of working copy. It's all human readable text, but if you mess it up you will have problems.
- On commit, CVS runs *diff* to produce list of differences between old and new version. Only these changes are stored in the repository.
- On update, CVS runs *patch* to apply the changes.

Binary files

- diff does not work for binary files, hence CVS can only store complete file each time – wastes diskspace
- Must use:
  `cvs add -kb binaryfile`
- If you forget the flag, file will be corrupted.

Advanced CVS: tags

- Can specify file revisions by number or by date:
- `cvs update -D “2003-12-25 7:05:00 GMT”`
- `cvs update -r 1.5 file`
- Tagging provides one name for all files at a certain 'snapshot' in time.
- `cvs -q tag Release-1-0`
- `cvs update -r Release-1-0`

Subversion

- The next generation of CVS
- Usage is almost identical
- More features, less gotchas
  - Directories
  - Binary files
- Download from *http://subversion.tigris.org*
- RapidSVN - GUI
- Free book at *http://svnbook.redbean.com*

Questions?

Documenting Code

Documenting Code

- Makes sense to document **in** the code, so docs will be kept up to date with code.
- Particularly important to document public APIs.
- If we write comments in a standard format, we can use a tool to extract them and produce docs.

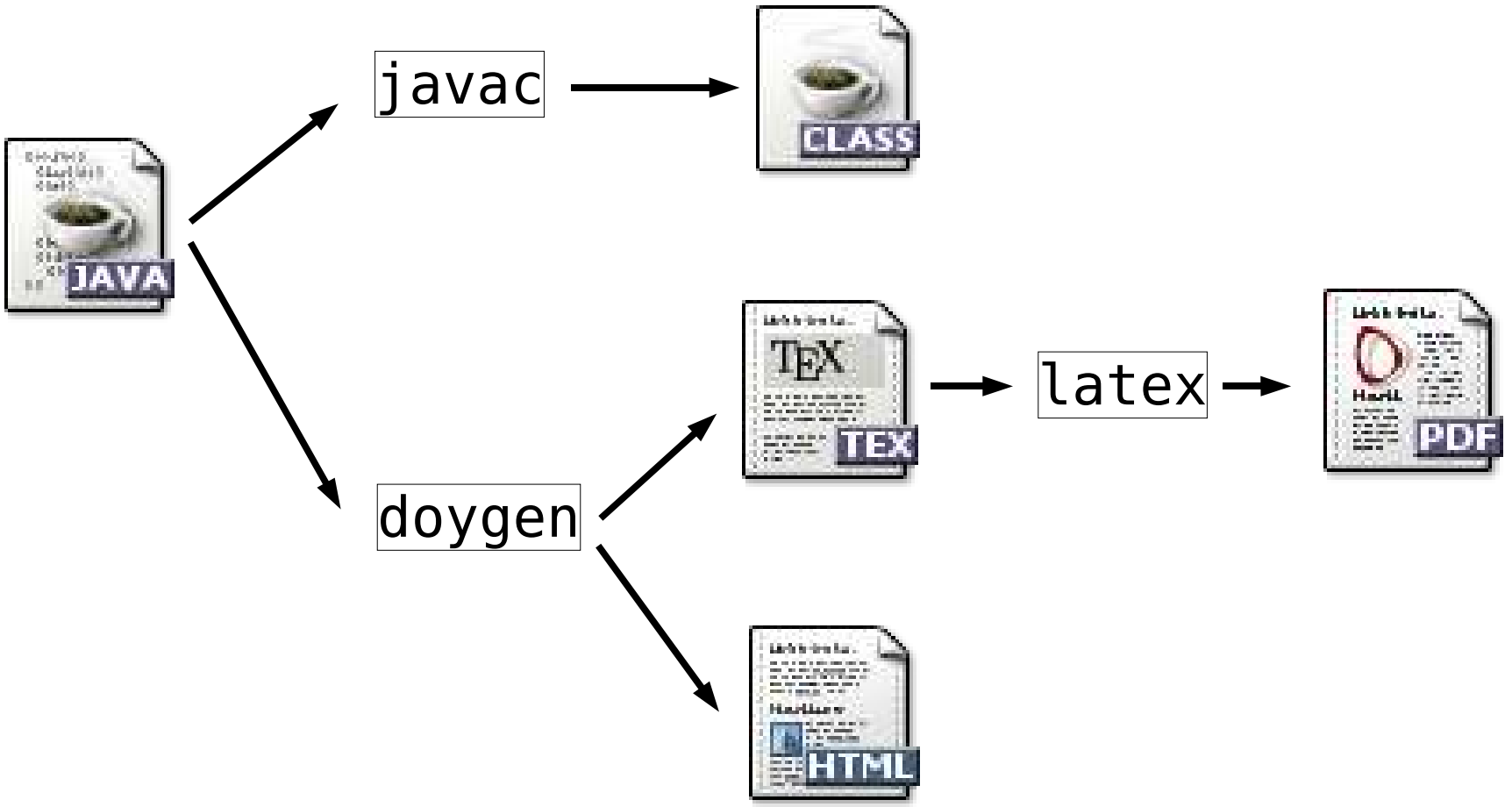
Tools

- Javadoc
  - supplied with Java
  - popular, because official Java docs use it
  - Java only
- Doxygen
  - backwards-compatible with Javadoc
  - extends Javadoc language
  - C++, Java and more
  - more output formats
  - *www.doxygen.org*

Using Doxygen

- Generate template Doxyfile:
  ```
  $> doxygen -g
  ```
- Edit Doxyfile:
  ```
  $> emacs Doxyfile
  ```
- Produce docs:
  ```
  $> doxygen
  ```


Work flow



Example Doxyfile options

```
PROJECT_NAME           = Animals
PROJECT_NUMBER         = 1
OUTPUT_DIRECTORY       = docs
OPTIMIZE_OUTPUT_JAVA   = NO
JAVADOC_AUTOBRIEF      = YES
GENERATE_HTML          = YES
GENERATE_LATEX         = YES
GENERATE_RTF           = NO
CLASS_DIAGRAMS         = YES
```

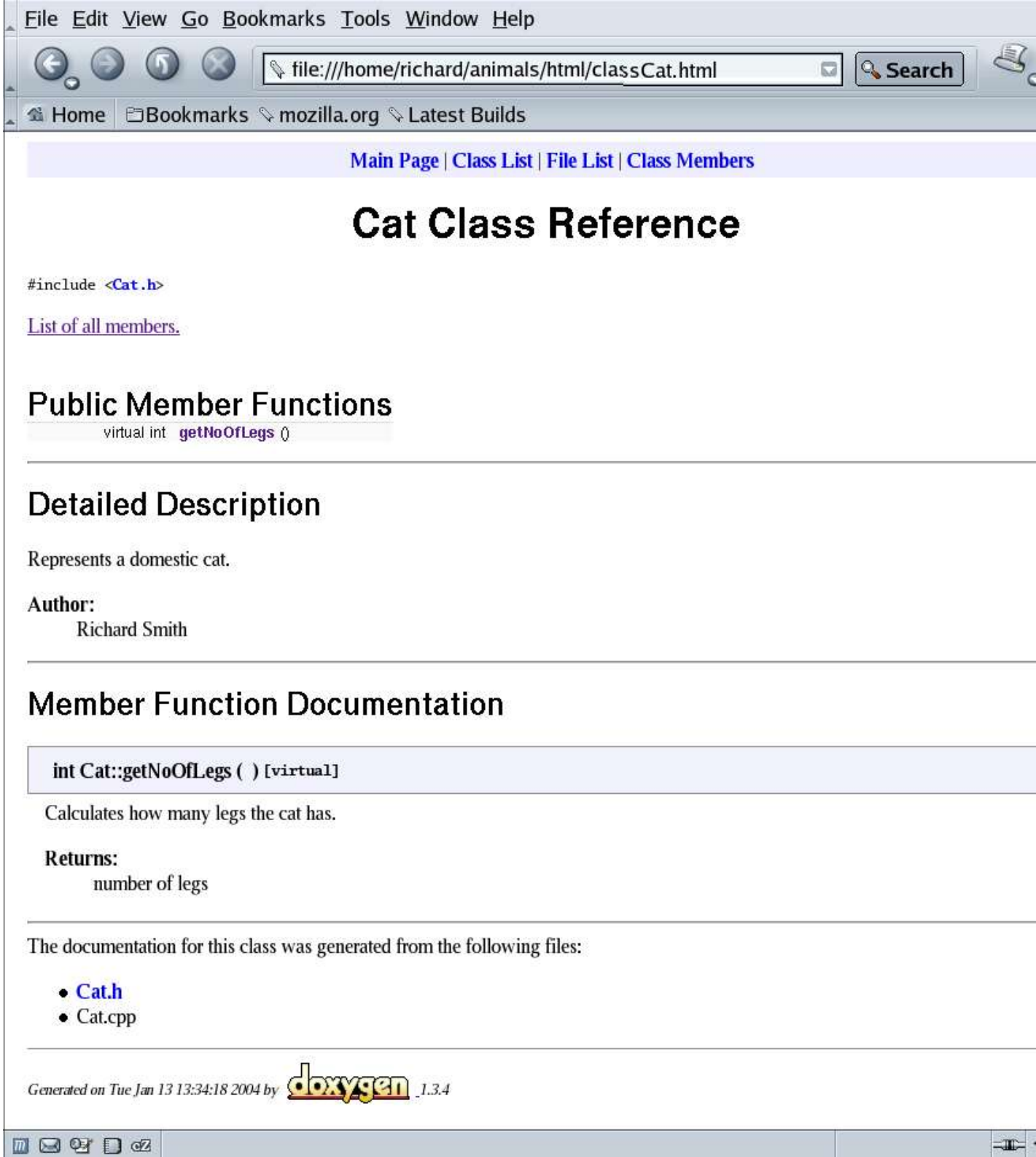

Doxygen Example

```
/**
 * Represents a domestic cat.
 * @author Richard Smith
 */
class Cat: public Animal
{
 public:

/**
 * Calculates how many legs the cat has.
 * @return number of legs
 */
  virtual int getNoOfLegs();
};
```

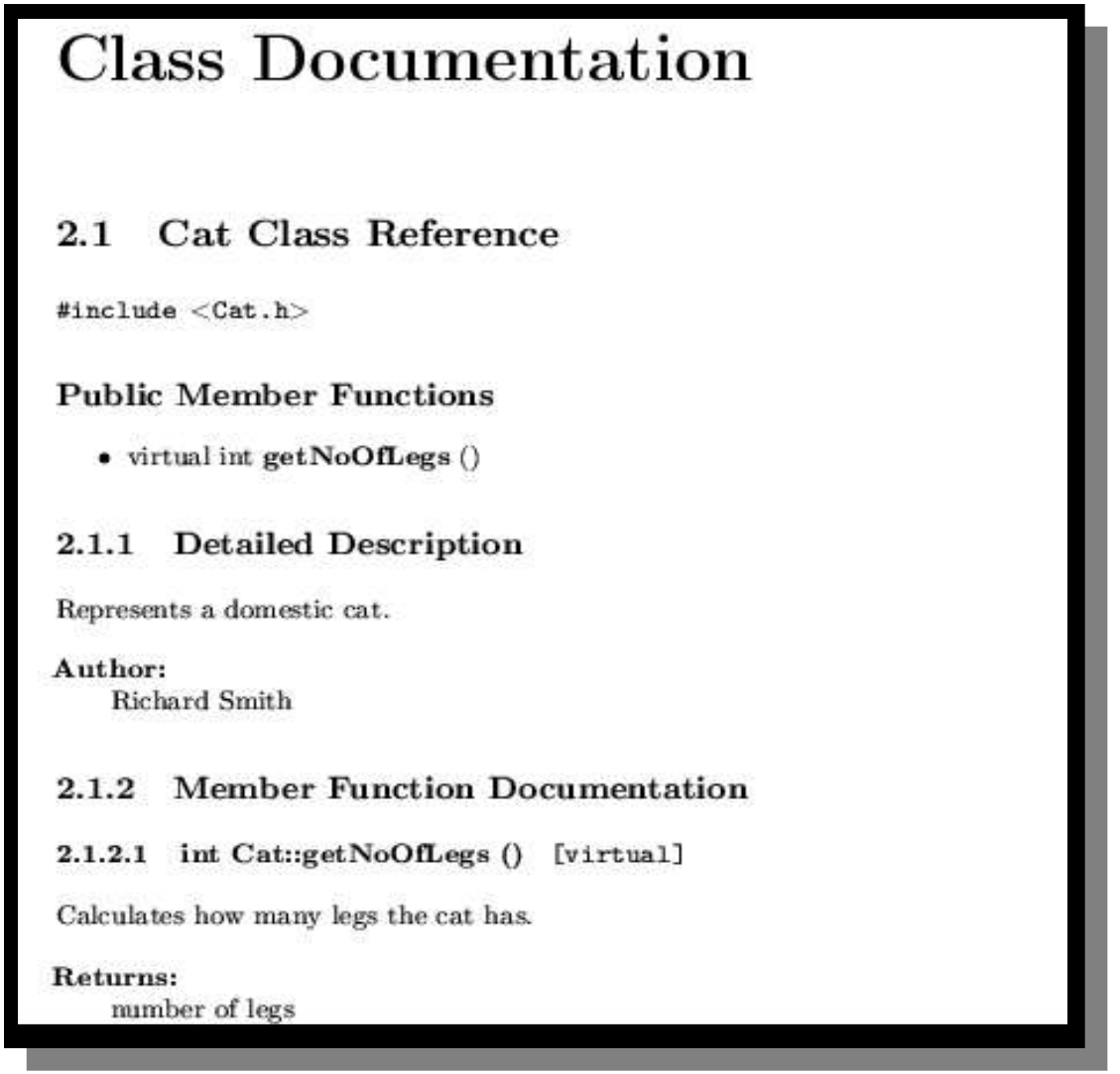

HTML Output

```
$> mozilla html/index.html
```



Latex output

```
$> cd latex
$> make refman.pdf
$> xpdf refman.pdf
```



Class Documentation

2.1 Cat Class Reference

#include <Cat.h>

Public Member Functions

- virtual int **getNoOfLegs** ()

2.1.1 Detailed Description

Represents a domestic cat.

Author:
Richard Smith

2.1.2 Member Function Documentation

2.1.2.1 int Cat::getNoOfLegs () [virtual]

Calculates how many legs the cat has.

Returns:
number of legs

Doxygen tags

- @author – can have multiple authors
- @return – what is returned by function
- @param – one for each function parameter
- @version
- @since – the version this function was introduced
- @throws - exceptions
- @see – other place to look for docs

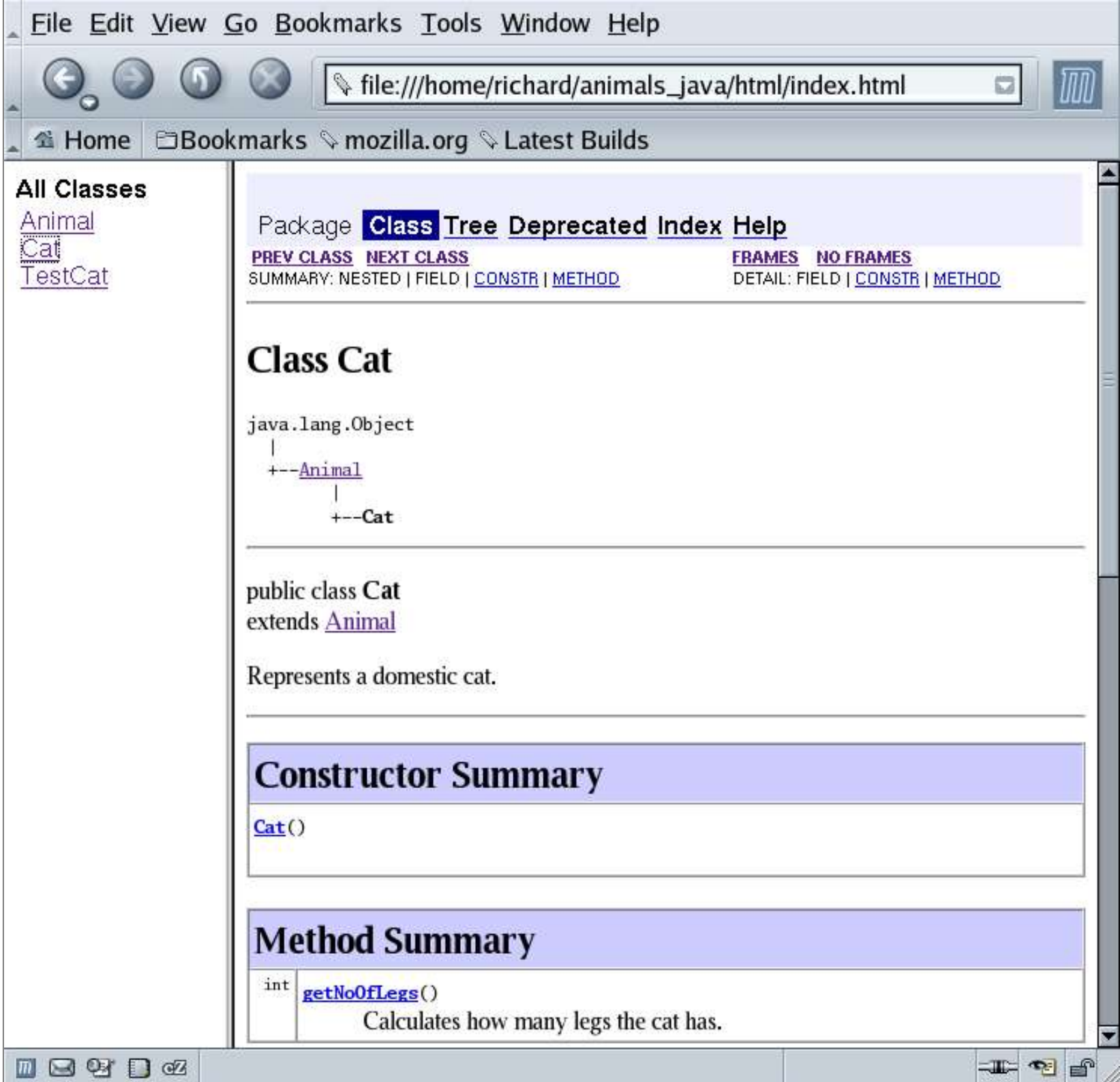
Using Javadoc

- ```
  $> javadoc -package -d html *.java
  $> mozilla html/index.html
  ```
- This shows methods/variables at package level. Can also show only private or public levels.
- Can specify package name rather than source files.


  ```

Javadoc Output

```
$> mozilla
html/index.html
```



Documenting – Report Writing

Word processors

- Binary file format – no version control – only one person can edit document – bottleneck
- Force you to worry about layout rather than content
- Cannot assume users have word processor – not suitable for doc distribution (but you can convert to PDF…)
- If using Microsoft Word, all developers must use same version and OS, else interchange may not be reliable

Text mark-up languages

- So we use plain text, with semantic 'mark-up tags'.
- The more detailed the mark-up, the more output options we have, but the more difficult it is to write.
- In order of complexity: HTML, LaTeX, DocBook
- Can always convert to less complex, but not to more complex

LaTeX

- **Excellent guide you should read** *ftp://ftp.funet.fi/pub/TeX/tex-archive/info/lshort/english/*
- Recommend use *pdflatex* version

example.tex

```
$> pdflatex example.tex
```

example.pdf

Simple document

```
\documentclass[a4paper,12pt]{report}

% this is a comment

\author{Richard Smith}
\title{My report}

\begin{document}
\maketitle

\chapter{A chapter}
\section{A section}
\subsection*{A subsection}
Because of the *, this section is not numbered.

Another \emph{important} paragraph\footnote{with
footnote}

\end{document}
```


Latex output

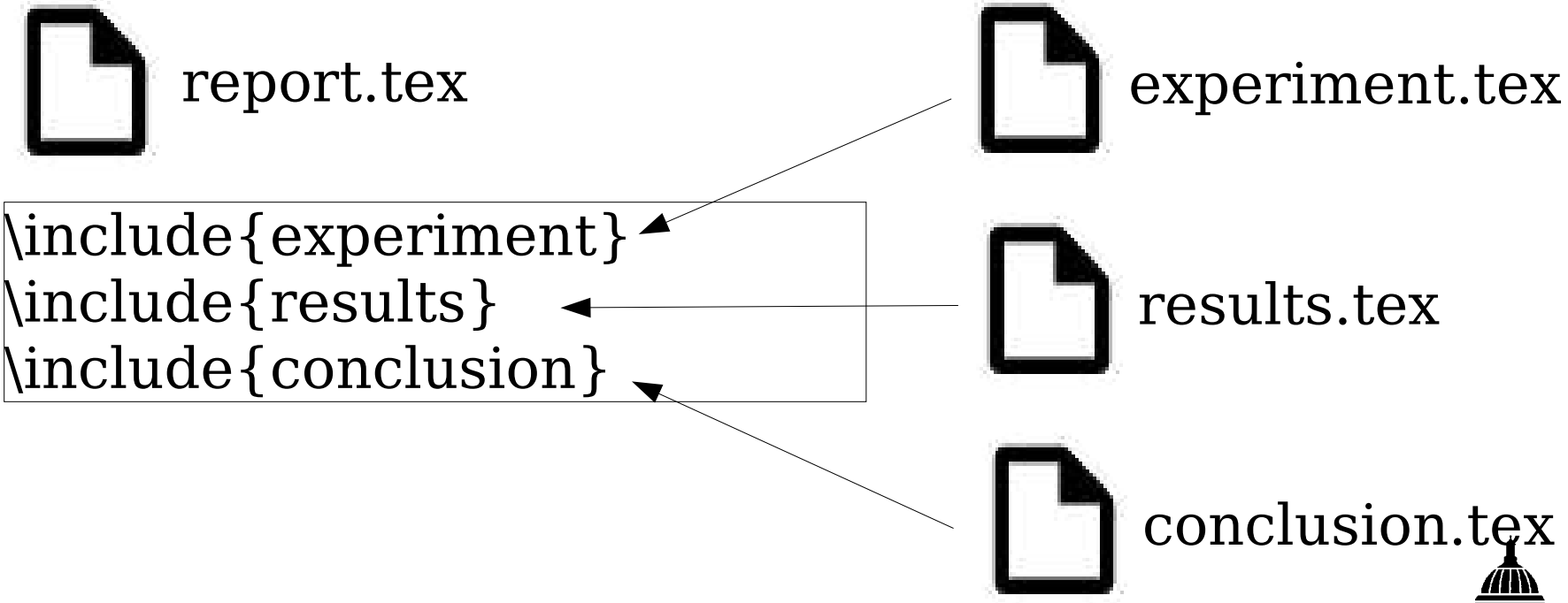
See demo

Graphics

- *Preamble:*
  ```
  \usepackage[pdftex]{color,graphicx}
  ```
- ```
  \begin{figure}[b]
  \begin{center}
  \includegraphics[width=7cm]{filename.png}
  \caption{My picture!}
  \end{center}
  \end{figure}
  ```
- *Also useful:*
  ```
  \tableofcontents
  \listoffigures
  ```
  ```

Multiple files

- Can break a large document up into separate files



Bibliography

- Will automatically generate references in text, e.g. [1] and then numbered biblography at end

```
We see in \cite{bhatti}.

\begin{thebibliography}{99}
\bibitem{bhatti}Bhatti S. N., Crowcroft J., 2002,
QoS Sensitive Flows: Issues in IP Packet
Handling, IEEE Internet Computing
```

- For more advanced system, refer to *bibtex* package

Cross references

```
\chapter{Animals}
\section{Rats}
\label{rats}
Rats are furry.

\section{Cats}
Cats like to eat rats.  For more
information, see Section \ref{rats} on
page \pageref{rats}.
```


Debugging - logging

Debugging

- Unit tests help you find bugs, but how do you then fix them?
- Debugger
  - allows you to pause execution at any point
  - view/change variables
  - single step through flow of execution
- But
  - doesn't help you debug released software that is running on other people's machines
  - is often 'overkill'

Logging

- Simply output important variable values and flow of execution points during execution
-

```
int legs=5;

public int getNoOfLegs(){
        System.out.println("getNoOfLegs() called");
        System.out.println("legs = "+legs");
        return legs;
    }
```

Logging tools

- Drawback of simple method – you have to comment out print statements and recompile to turn off logging
- So then you start to do
  `if(logging==true) print ....`
- Log4j does this for you, plus adds a lot of logging flexibility that your sysadmin users will appreciate

javac CatLogged.java -classpath .:log4j-1.2.8.jar

Example - CatLogged.java

```
import org.apache.log4j.Logger;

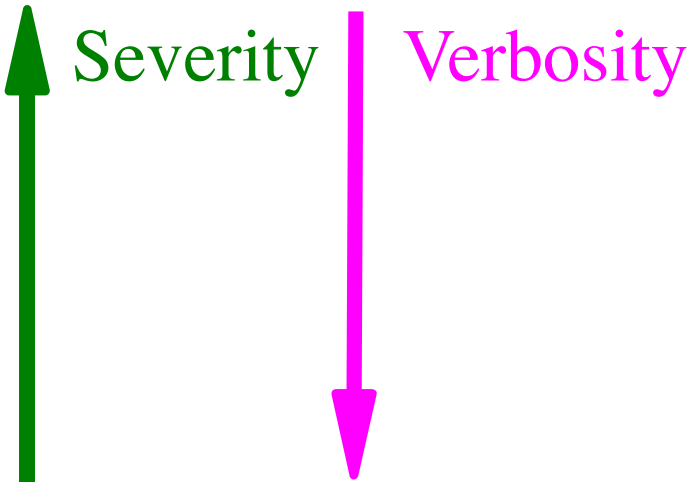
public class CatLogged extends Animal{

   static Logger logger = Logger.getLogger
(CatLogged.class.getName());
   int legs=5;

   public int getNoOfLegs(){
      logger.warn("getNoOfLegs() called");
      logger.debug("legs ="+legs);
      return legs;
   }
}
```


Log levels

- We only see output from the current level and *higher* levels
- FATAL
- ERROR
- WARN
- DEBUG



log4j.properties

```
log4j.rootLogger=debug, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern
   =%5p [%t] (%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log

log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

Output

- **Warn** level:

```
GetNoOfLegsCalled()
```

- **Debug** level:

```
getNoOfLegsCalled()
legs=5
```