Managing recompilation

- What happens if a source file is changed?
  - Possibly need to recompile all the files that referenced it
- How to do this?
  - IDE: built-in
  - So far: by hand
    - Call `javac on out-of-date source files, maybe re-jar`
    - But: tedious, error prone
  - Tool-based approach: make a tool for it!

make

- `make` is a great tool that manages any kind of process with dependencies
- A `Makefile` describes rules for when something depends on something else, and what to do to make it up-to-date
  - based on file modification times stored with every Unix file
- Invoking `make` then runs these rules to decide what, if anything, needs to be done to bring things up-to-date

Dependencies

- **Makefile** includes lines of the form
  
  ```
  target.. : source..
  ```

  Means that each target depends on each source
  - If any of the sources are modified, then all the targets are considered out-of-date
- **Example:**
  ```
  main.class: main.java
  ```

Actions

- For each dependency, can add an action to perform to bring the target(s) up-to-date
  - Action is a series of shell command lines
    - each line must start with a `tab`
    - use `/bin/sh` syntax
- **Example:**
  ```
  main.class: main.java
  javac main.java
  ```

Invoking make

- `make target..`
  - uses `Makefile` in current directory to bring one or more targets up to date, using their actions
  - does nothing if all targets up to date
  - if omit target arguments, then rebuild the first target in `Makefile`
    - the default target
- **Example:**
  ```
  > make main.class
  javac main.java
  >
  ```

Controlling output

- By default, `make` prints out each action it performs
- Can disable printing an action by prefixing it with `!`
- **Example:**
  ```
  main.class: main.java
  $echo Compiling main.java
  $javac main.java
  > make main.class
  Compiling main.java
  >
Dependency patterns

- Often have a simple rule over all files with certain naming patterns
  - Can use % in the target and source
  - Rule applies to any real targets and sources where % is replaced by the same thing on both sides
- Example:
  - %.class: %.java
  - Means that %.class depends on %.java

Actions for patterns

- Actions for dependency patterns need to have patterns too
  - $@: the target
  - $^: the source(s)
  - $$: the thing matched by % in the rule
- Example:
  - %.class: %.java
    @echo -n "compiling class $\" $ *
    @echo "$\$" to $\"
  - javac $\$

Dependency trees

- One target can depend on another target, ad nauseum
  - Dependency rules form a DAG (directed acyclic graph)
  - make figures out how to rebuild a target by first making sure its sources are up-to-date, which may cause make to first rebuild them, recursively

Example dependency tree

- %.class: %.java
  - javac $^ 
  - main.jar: main.class helper.class
    - jar cfv $@ $^ 
  - install: main.jar
    - cp $^ $(HOME)/bin 
  - > make install
    - javac main.java
    - javac helper.java
    - jar cfv main.jar main.class helper.class 
    - cp main.jar /homes/jws/mylogin/bin 

Makefile variables

- Can define variables in Makefile\$s, and use them in rules and actions
  - VARNAME = REPLACEMENT...
  - Referenced using $\{ VARNAME $\}
- Example:
  - JAVAC\_FLAGS = -g 
  - %.class: %.java
    - @echo "compiling class $\" 
    - javac $(JAVAC\_FLAGS) $\$

Substitutions in make vars

- Can do replacements in variables
  - $\{ VAR: oldPat= newPat $\}
  - oldPat and newPat can contain %
  - match each word in $\{ VAR $\} against oldPat, where % can match anything
  - replace matches with new
  - if new contains %, substitute with what % matched
  - Good for adjusting extensions, prefixes
Examples of substitutions

SRCS = A.java B.java C.java
OBJS = $(SRCS:%.java=%.class)
default: $(OBJS)

INSTALL_DIR = $(HOME)/bin
INSTALLED_OBJJS = \$(OBJS:=%$(INSTALL_DIR)/%)
$(INSTALL_DIR)/%.class: %.class
cp ^ $@
install: $(INSTALLED_OBJJS)

Make quiz

- Extend Makefile so that "make clean" removes all .class files
- Add a rule so that I can say "make foo.java.ps", for any foo.java, to format my Java source file using enscript -2r into a nicely formatted .ps file
- Add a rule to put all my .class files into a single .jar file
- Add a variable defining all the .java files in my application, and only clean, format, and archive those files