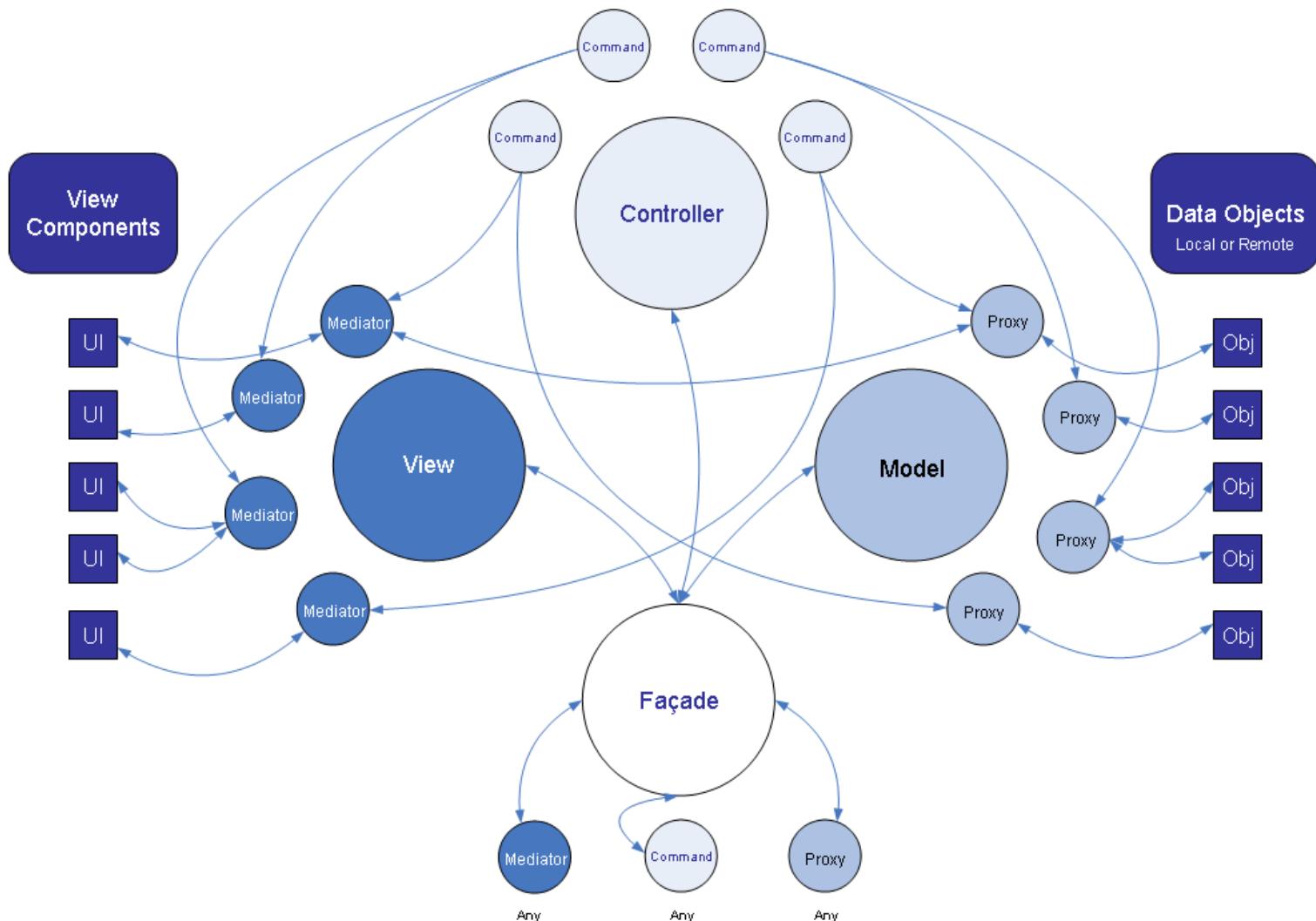


CS209 A

Week 13

HARD QUIZ

Software System Design: Patterns



The Pattern Concept

- History: Architectural Patterns
- Christopher Alexander
- Each pattern has
 - a short *name*
 - a brief description of the *context*
 - a lengthy description of the *problem*
 - a prescription for the *solution*

A Pattern Language

Towns · Buildings · Construction



Christopher Alexander

Sara Ishikawa · Murray Silverstein

WITH

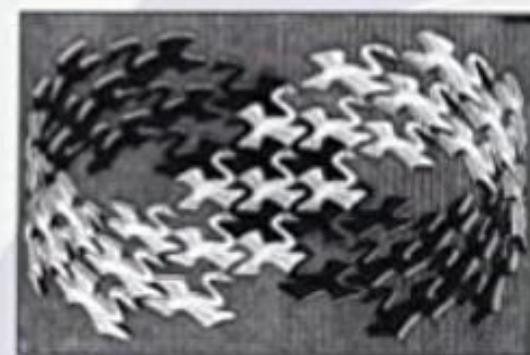
Max Jacobson · Ingrid Fiksdahl-King

Shlomo Angel

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

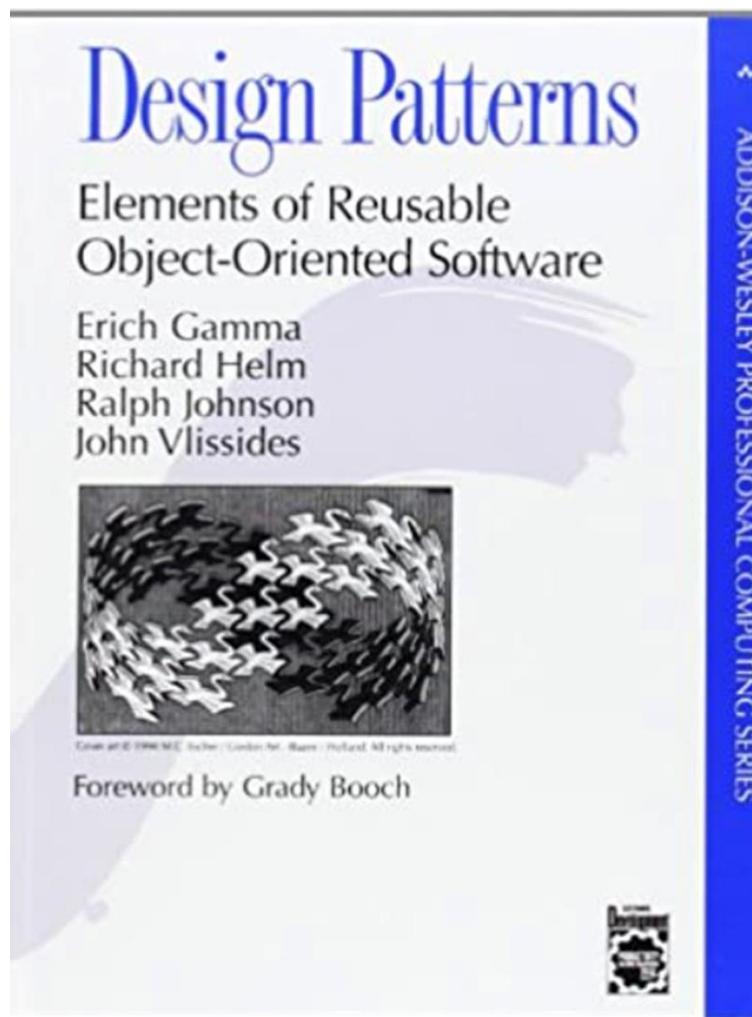


© 1994 Addison Wesley Longman, Inc. All rights reserved.

Foreword by Grady Booch



Influential Book



Short Passages Pattern



Do you like this kind of building?

Short Passages Pattern

Context

"...Long, sterile corridors set the scene for everything bad about modern architecture..."

Problem

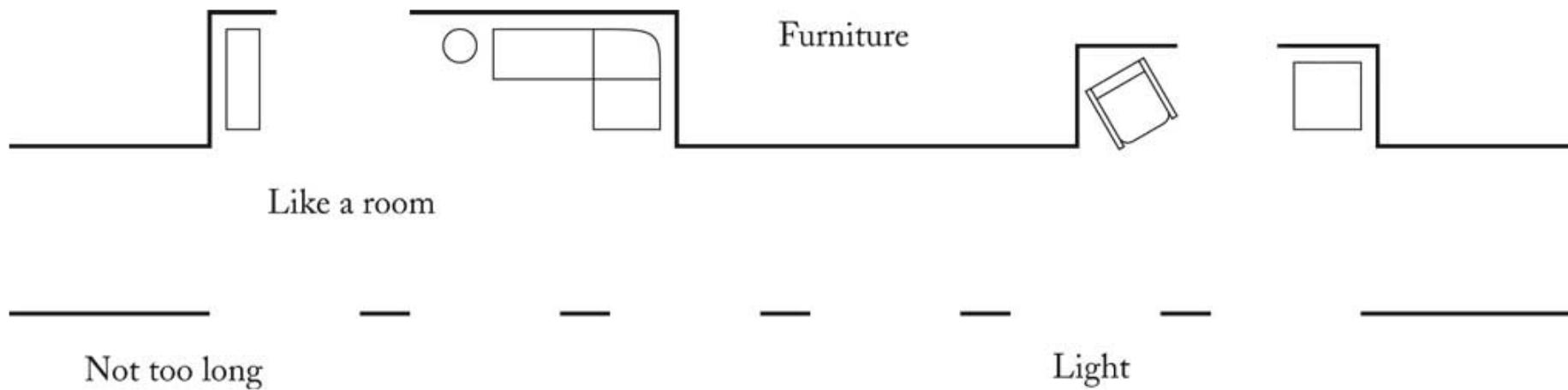
a lengthy description of the problem, including

- a depressing picture
- issues of light and furniture
- research about patient anxiety in hospitals
- research that suggests that corridors over 50 ft are considered uncomfortable

Short Passages Pattern

Solution

Keep passages short. Make them as much like rooms as possible, with carpets or wood on the floor, furniture, bookshelves, beautiful windows. Make them generous in shape and always give them plenty of light; the best corridors and passages of all are those which have windows along an entire wall.



Iterator Pattern

Context

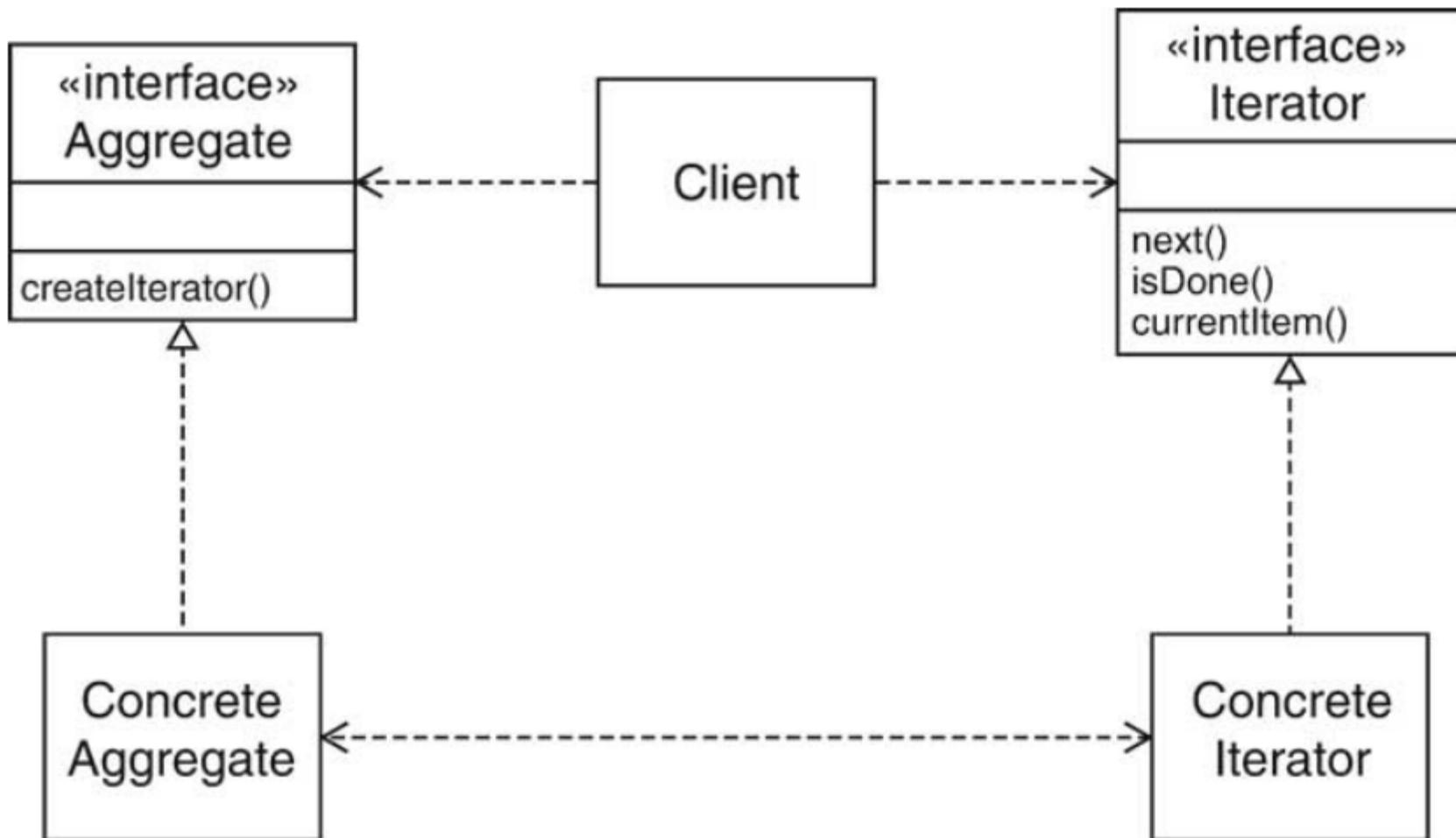
1. An aggregate object contains element objects
2. Clients need access to the element objects
3. The aggregate object should not expose its internal structure
4. Multiple clients may want independent access

Iterator Pattern

Solution

1. Define an iterator that fetches one element at a time
2. Each iterator object keeps track of the position of the next element
3. If there are several aggregate/iterator variations, it is best if the aggregate and iterator classes realize common interface types.

Iterator Pattern



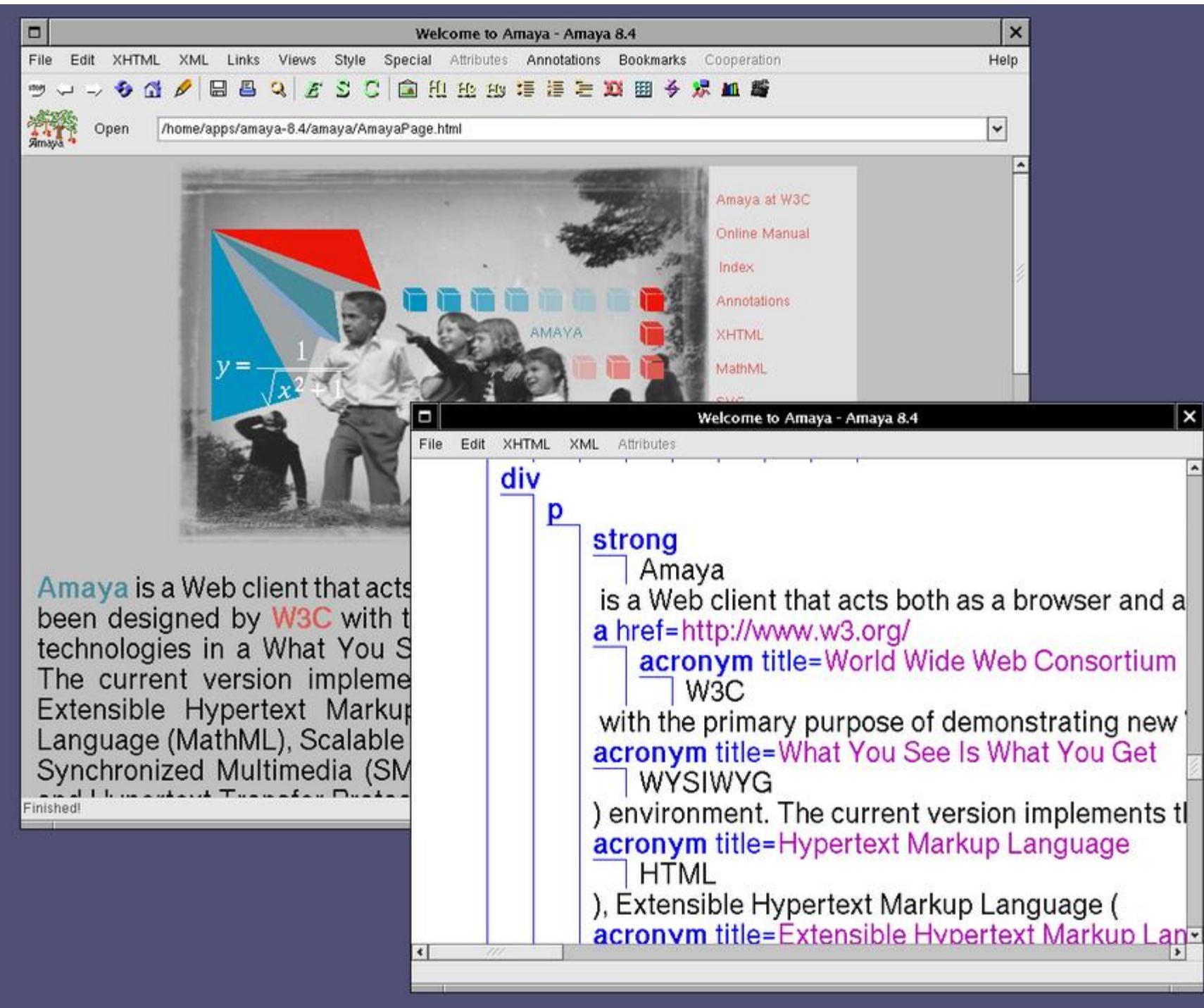
Iterator Pattern

- Names in pattern are *examples*
- Names differ in each occurrence of pattern

| Name in Design Pattern | Actual Name (linked lists) |
|------------------------|---|
| Aggregate | List |
| ConcreteAggregate | LinkedList |
| Iterator | ListIterator |
| ConcreteIterator | anonymous class implementing ListIterator |
| createIterator() | listIterator() |
| next() | next() |
| isDone() | opposite of hasNext() |
| currentItem() | return value of next() |

Model/View/Controller

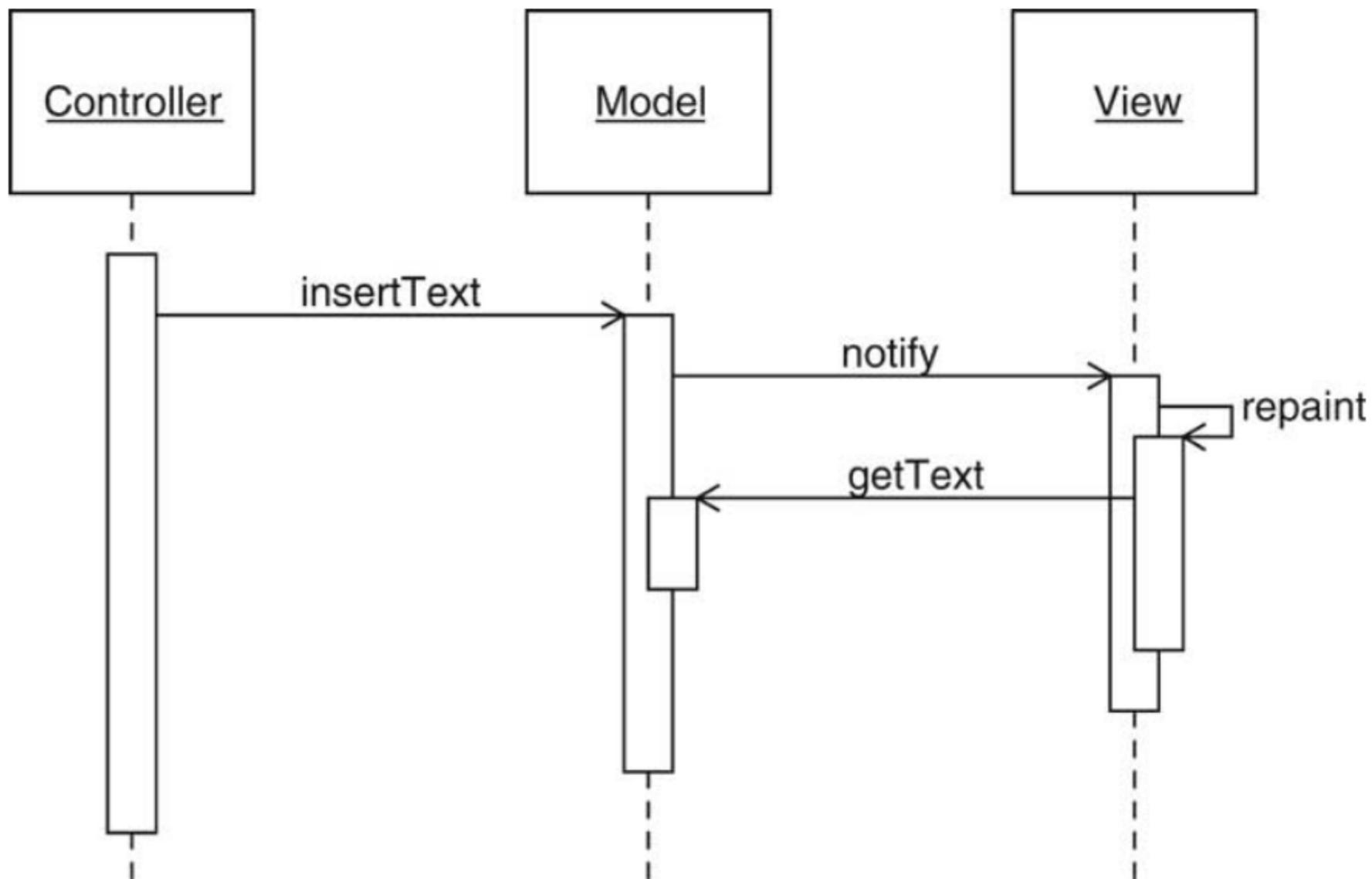
- Some programs have multiple editable views
- Example: HTML Editor
 - WYSIWYG view
 - structure view
 - source view
- Editing one view updates the other
- Updates seem instantaneous



Model/View/Controller

- Model: data structure, no visual representation
 - Views: visual representations
 - Controllers: user interaction
-
- Views/controllers update model
 - Model tells views that data has changed
 - Views redraw themselves

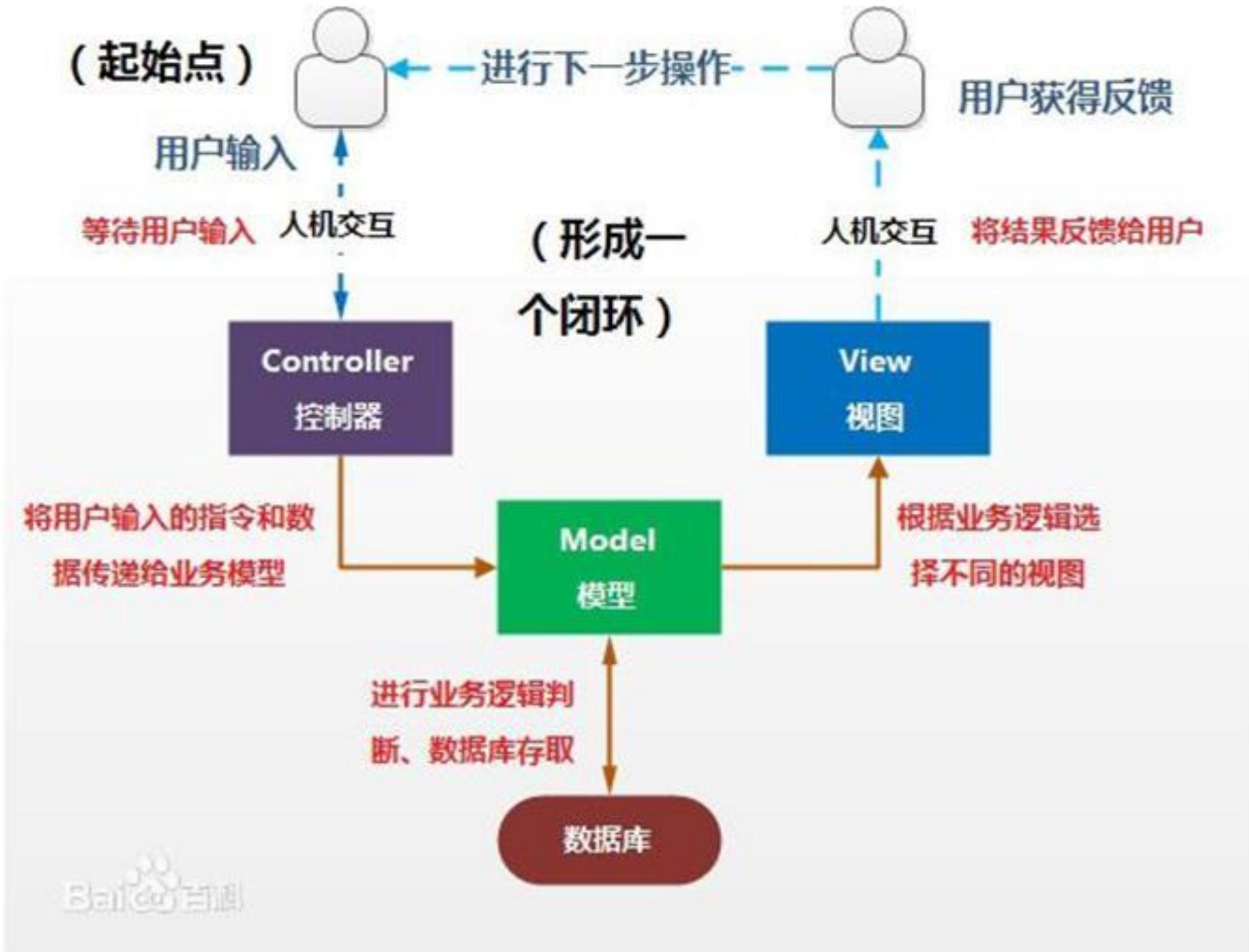
Model/View/Controller



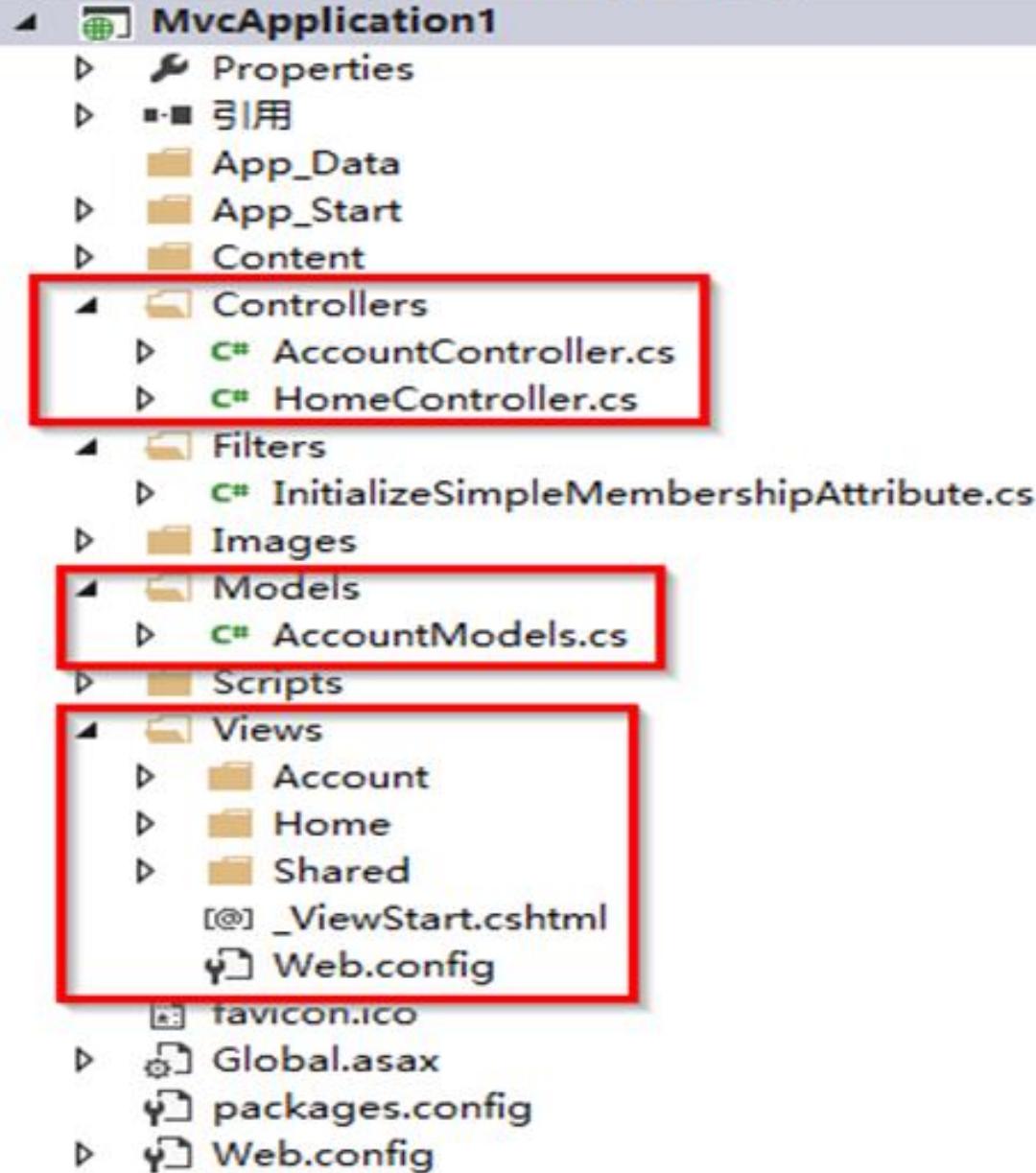
MVC



- V) view视图：是指用户看到并与之交互的界面。
- 比如由html元素组成的网页界面，或者软件的客户端界面。MVC的好处之一在于他能为应用程序处理很多不用的视图。在视图中其实没有真正的处理发生，它只是一个作为输出数据并允许用户操纵的方式。
- M) model模型：是指模型表示业务规则。
- 在MVC的三个部件中，模型拥有最多的处理任务。被模型返回的数据是中立的，模型与数据格式无关，这样一个模型能为多个视图提供数据，由于应用与模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。
- C) Controller控制器：
- 是指控制器接收用户的输入并调用模型和视图去完成用户的需求，控制其本身不输出任何东西和任何处理。它只是接收请求并解决调用哪个模型构建去处理请求，然后再确认用哪个视图来显示返回的数据。



解决方案“MvcApplication1”(1 个项目)



Software Development Process

Writing code is one thing. But...

- **PROJECTS REQUIRE MORE**

- Documentation
- Organization
 - Code
 - Build
 - Integration
- Testing
- Deployment

In general, and with any language, a project is more than writing one program and there is a whole eco-system around it. This will be a brief overview, mostly to provide ideas and topics to search on the web.

Documentation

**Everyone wants documentation
Everyone hates writing it**

You are probably already somewhat familiar with javadoc and were asked to use it in your project... Uses specially formatted comments in the code which can then be used to generate a document describing the elements of the software system (classes, methods, packages and so forth).

See: <https://www.oracle.com/java/technologies/javase/javadoc-tool.html#javadocdocuments>

Software Organization



Organizing Code

In industry projects where there is a large code base, and perhaps the development team may change, there is a need for a lot of organization and methods.

Side notes:

- if you are interested in software projects, there is a famous book called "The Mythical Man-Month" by Fred Brooks. It's based on the experience on a project 50 years ago but, it's still quite relevant today.
- Another famous book is "Implementing Lean Software Development: From Concept to Cash" by Mary and Tom Poppendieck, which has a lot to say about software development strategies particularly in focusing on aspects that improve efficiency and avoid creating unnecessary work.

Organizing Code

```
public class  
MyCode {  
  
    ...  
    Lots of methods  
    ...  
  
    static void main()  
{  
}  
}
```

The "one big file with lots of methods" approach cannot work with several people working on the same project. You have to break everything into smaller classes with their own methods, calling methods from other classes.

Organizing Code

Class 1

Class 2

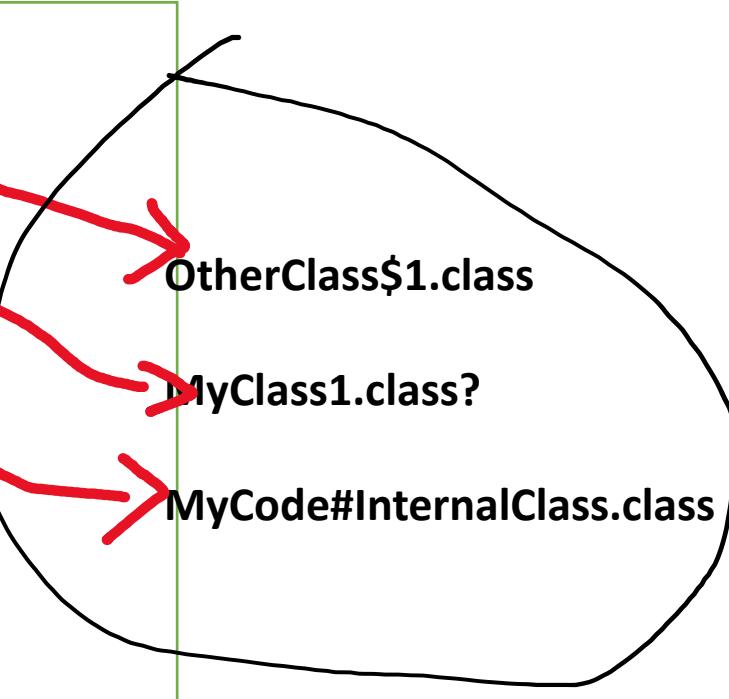
Class 3

```
public class  
MyCode {  
static void main()  
{
```

- Only one class contains a public static void main() methods. Other classes are public classes with constructors and methods, but no main().
- It's interesting to see what javac generates when applied to a single .java file. It won't always be a single .class file.

Javac

```
class OtherClass{  
}  
  
public class MyCode {  
    MyClass1 mc;  
    ...  
    class InternalClass{  
        ...  
    }  
    static void main() {  
    }  
}
```



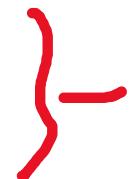
Classes outside the main one, as well as internal classes, will be turned into separate .class files. In fact (surprise) you have one .class per class.

If you are using another class, the JVM loader expects to find the corresponding .class ‘somewhere’ (it looks in the current directory, then the CLASSPATH)

Loading Classes

The loader will look in what is supplied with the Java runtime environment, by default it will look into the current directory or any directory or .jar file (more about .jar files later) that you specify.

- Looks for .class files in the current directory (folder)
- Or in directories specified in CLASSPATH
- Or with -cp
 - *directory1:directory2:....:*
 - *directory1;directory2;...;...*
- Special Directories:
 - “.” = current
 - “..” = parent



Or .jar files

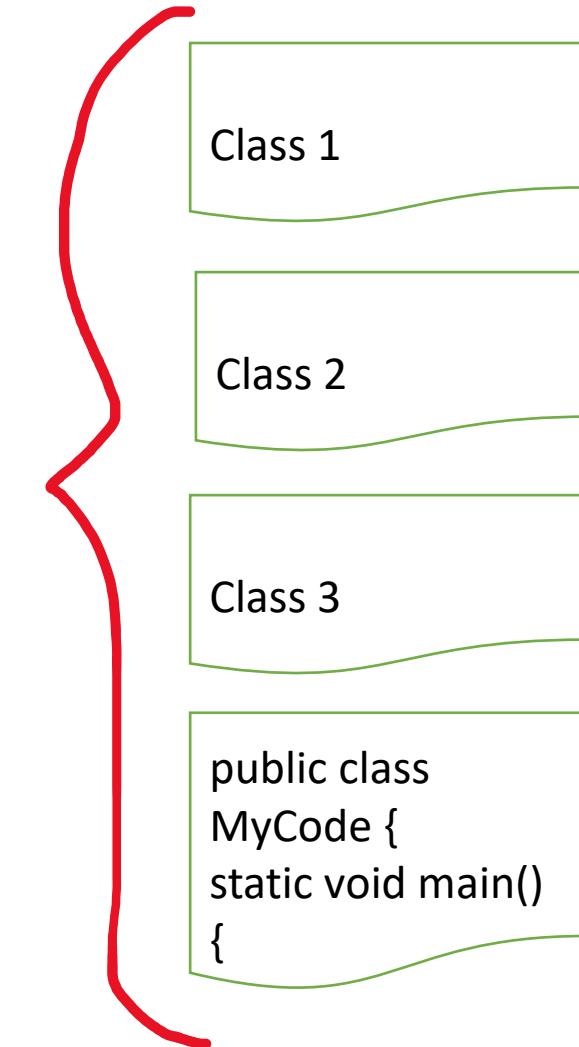
Packages

"Packages", as the name suggests, as for storing related software classes together.

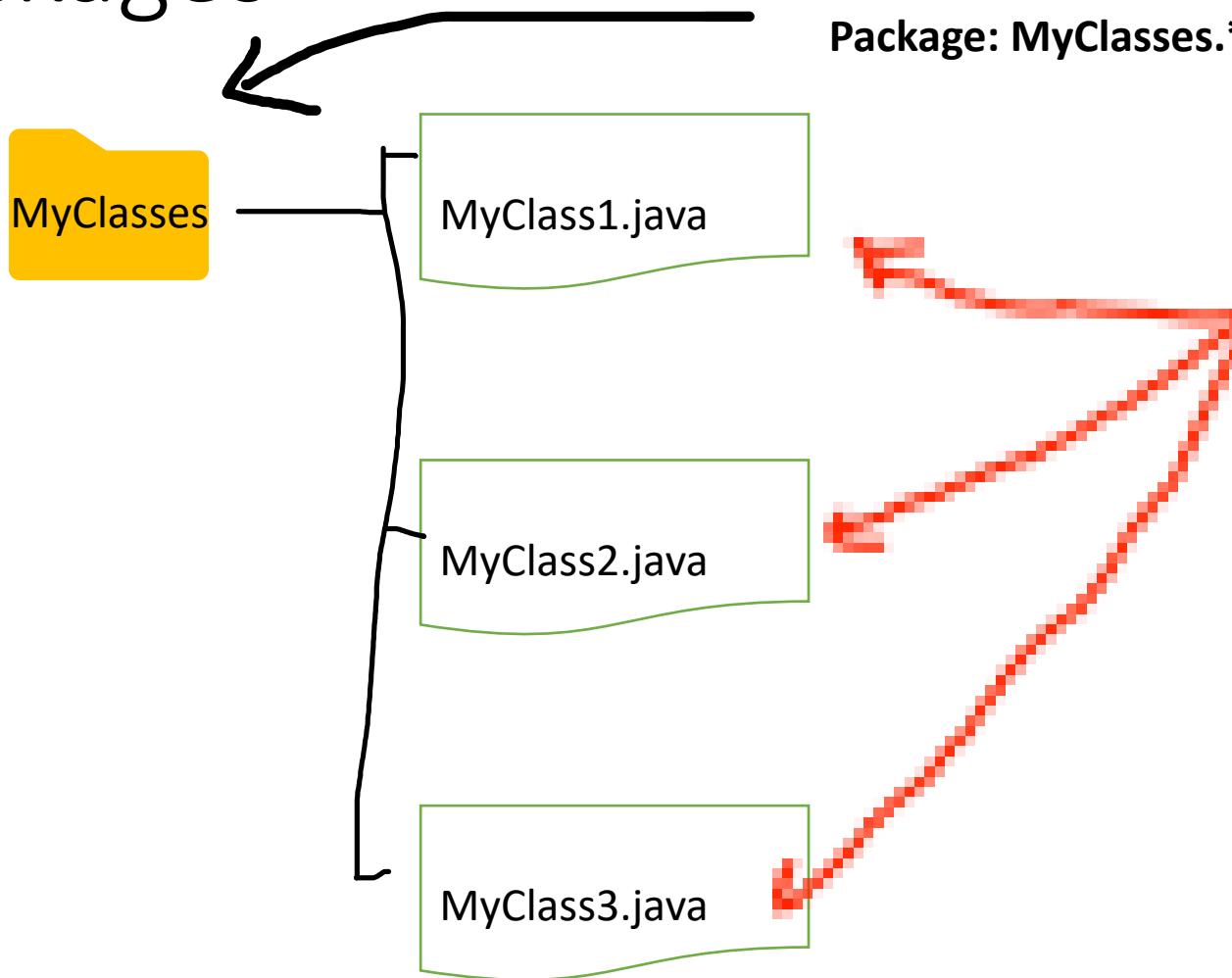


Packages

You want to use packages especially when you feel that the classes could be reused in a different software project.



Packages



In practice a package is nothing but a directory. You must name the package at the beginning of the files in the package.

```
import MyClasses.*;
```

```
public class MyCode {  
    static void main() {
```

Packages and naming

There is always a problem with naming: the possibility of a name conflict or giving something a name that already exists



To avoid name conflict, companies have been known to build package directories that look like their reversed website name, followed by whatever they want. By giving the full package name, you know exactly what you are talking about.

Software Build Tools

Apache Ant



Build tools

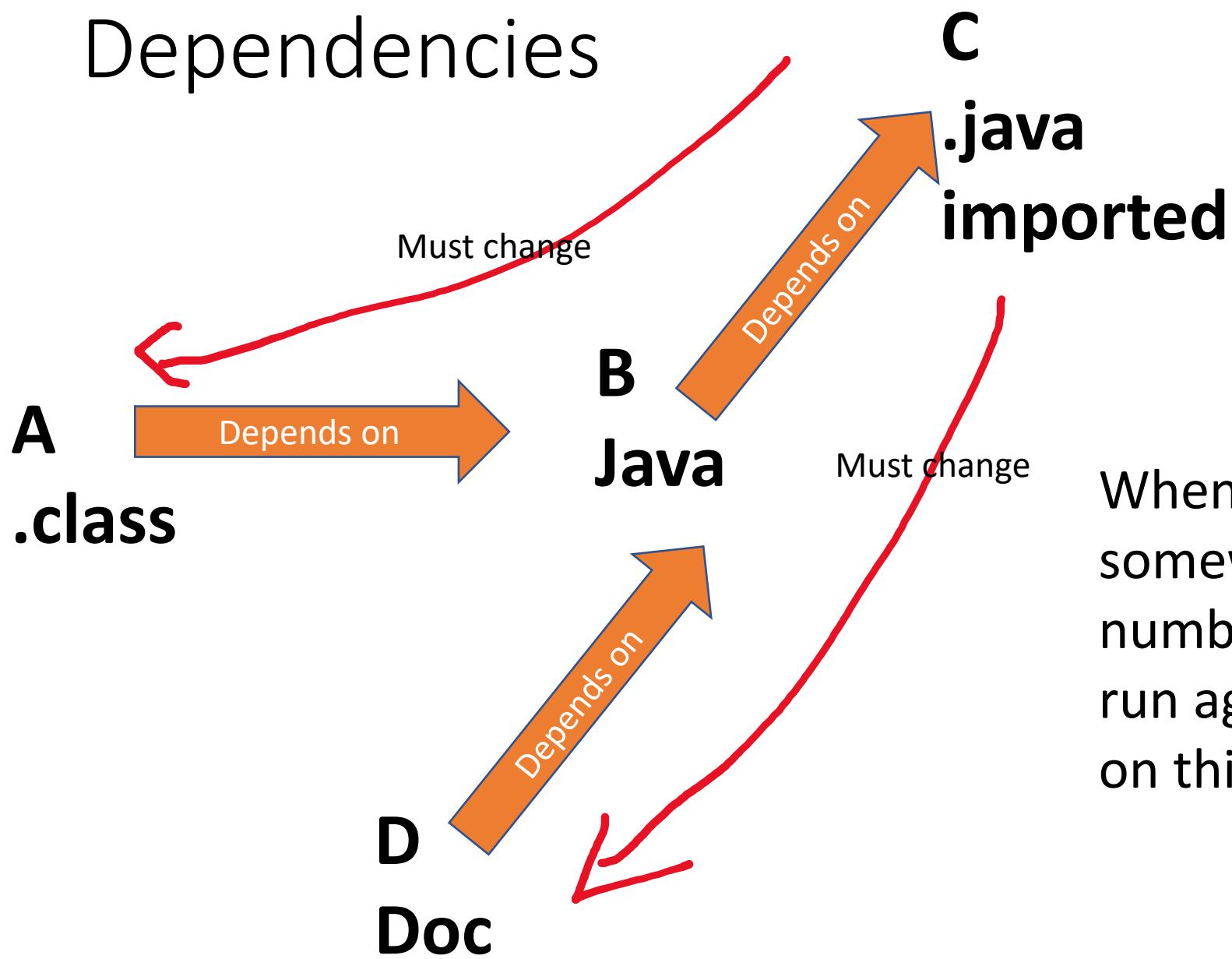
Bigger project = Many components
Dependencies...

Running javac isn't the only thing you have to do...

Before you compile the code, you must often extract it from a repository (more about this later). If javac is successful, you need to generate the doc, (with Javadoc tool is similar to compiling code - or in eclipse there is a wizard). You need to run (automated) tests, you need to prepare software for deployment...

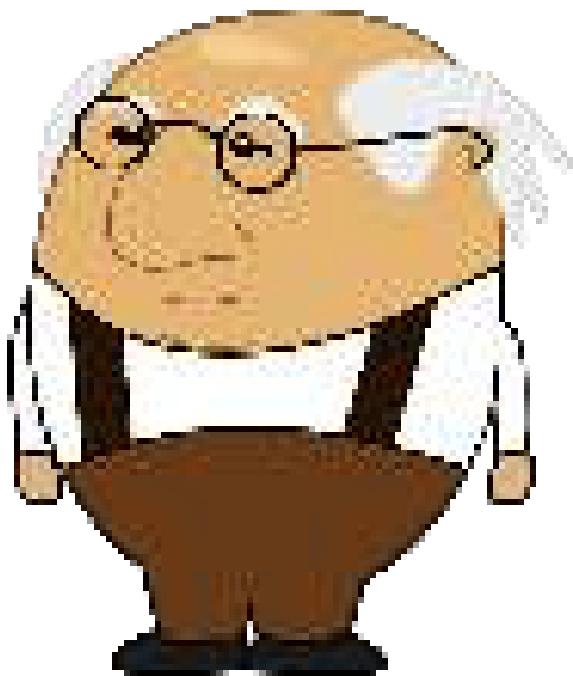
A lot of small, often boring tasks, depending on each other, that developers have tried to automate as much as possible.

Dependencies



When you change something somewhere, there are usually a number of tasks that must be run again on whatever depends on this component.

Build Tools: Make



- The grand-dad of build tools is make, still very much in use and very popular among C developers.
- You can use it with Java as well!

Other build tools appeared that were more specifically targeted at Java development.

As they were created when XML was all the rage, they usually rely heavily on XML configuration files saying what depends on what and what to do when something changes.



Apache Ant : Build.XML



```
<?xml version="1.0"?>
<project name="Hello" default="compile">
    <target name="clean" description="remove intermediate files">
        <delete dir="classes"/>
    </target>
    <target name="clobber" depends="clean" description="remove all artifact files">
        <delete file="hello.jar"/>
    </target>
    <target name="compile" description="compile the Java source code to class files">
        <mkdir dir="classes"/>
        <javac srcdir"." destdir="classes"/>
    </target>
    <target name="jar" depends="compile" description="create a Jar file for the application">
        <jar destfile="hello.jar">
            <fileset dir="classes" includes="**/*.class"/>
            <manifest>
                <attribute name="Main-Class" value="HelloProgram"/>
            </manifest>
        </jar>
    </target>
</project>
```

(from Wikipedia)

Apache Ant



Describes dependencies and tasks:

- Get source code out of source control
- Compile
- Run test
- Copy file
- And so on...

All common tasks correspond to an XML tags, with the detail provided as XML attributes.

Software Build Tools

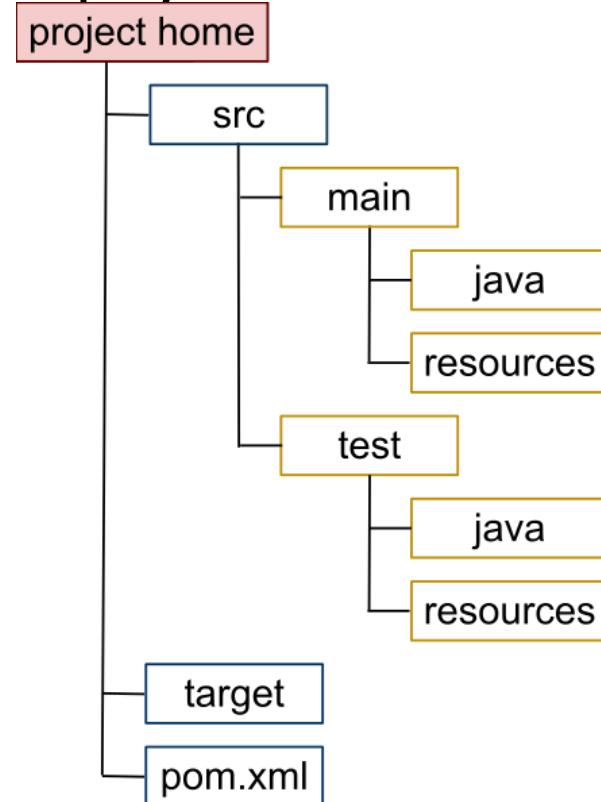
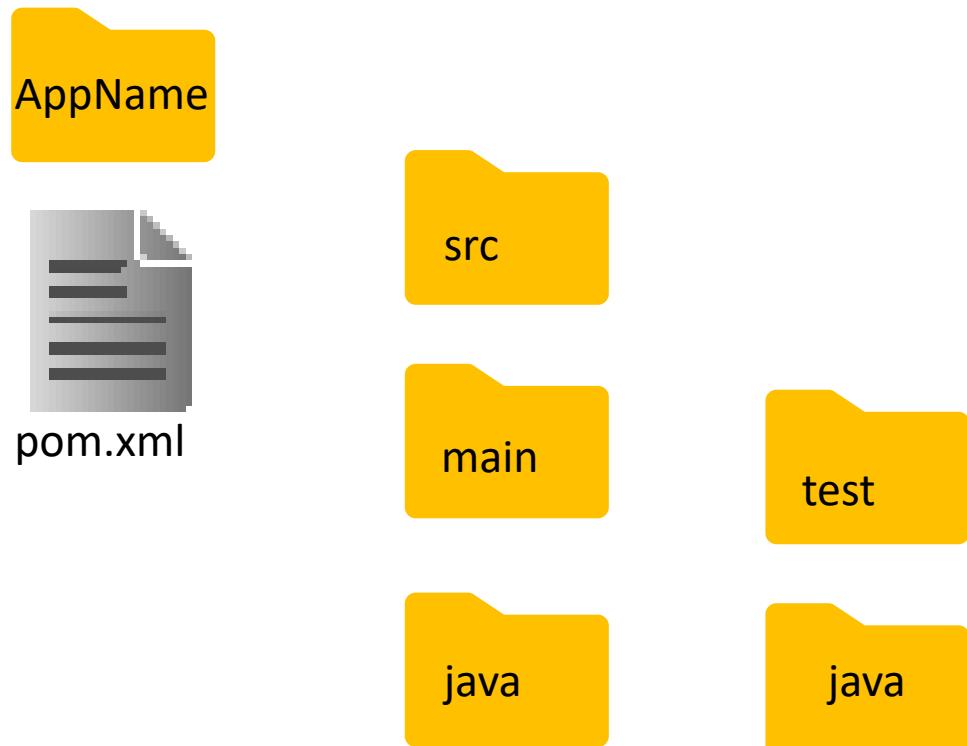


Project Object Model

pom.xml

maven

Maven is based on a structure that is always the same, and many common tasks are implicit. What isn't implicit is described in an XML file. “Convention over Configuration” philosophy...



- Maven handles dependencies and compilation/tool execution
- It is designed to be easy to use and provides pre-defined targets for things like compilation, packaging, deployment
- Unlike Ant it uses conventions which if followed don't need to be explicitly specified
 - For example: build order, running unit tests, and much more
- Only things that are different from the standard conventions need to be specified in the pom.xml file



Build Lifecycles

Most of Maven's functionality is in plug-ins. A plugin provides a set of goals that can be executed using the command mvn [plugin-name]:[goal-name]. For example, a Java project can be compiled with the compiler-plugin's compile-goal by running mvn compiler:compile.

The build lifecycle is a list of named *phases* that can be used to give order to goal execution. One of Maven's standard lifecycles is the *default lifecycle*, which includes the following phases:

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-test-sources
- process-test-resources
- test-compile
- test
- package
- install
- deploy





git



Source Control Systems

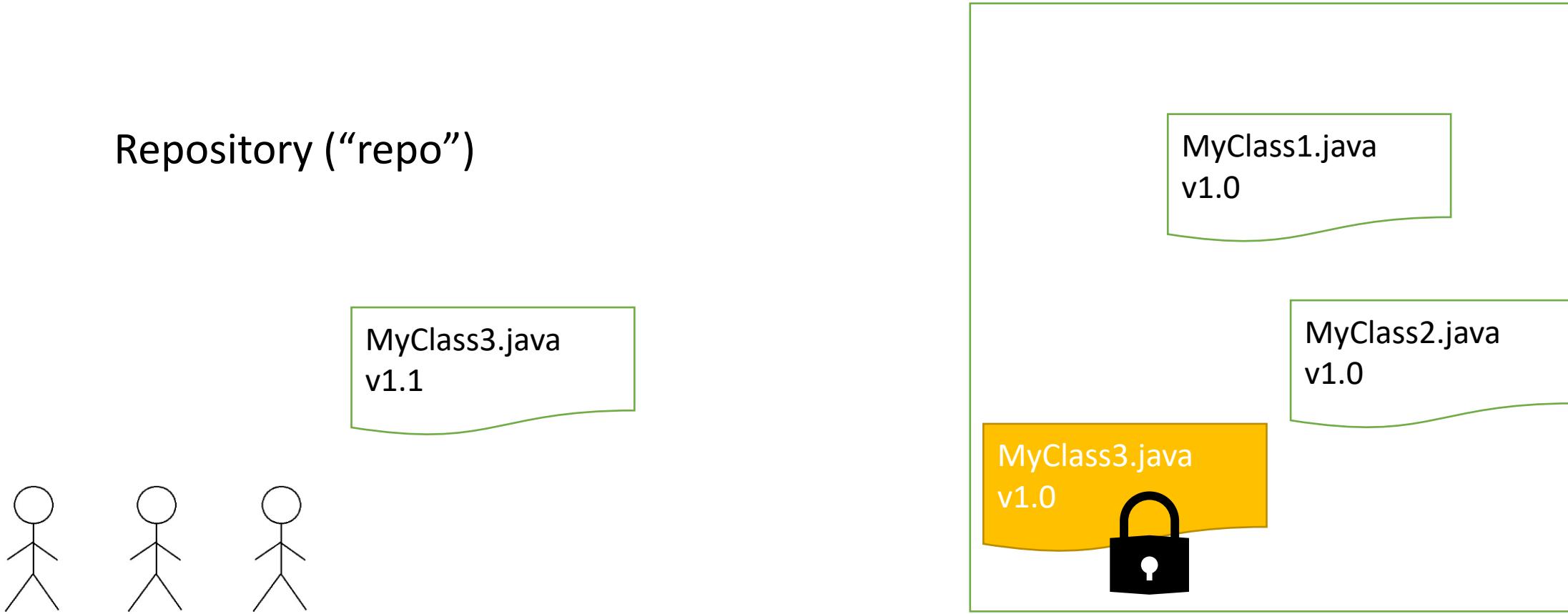


mercurial

Source Control Systems

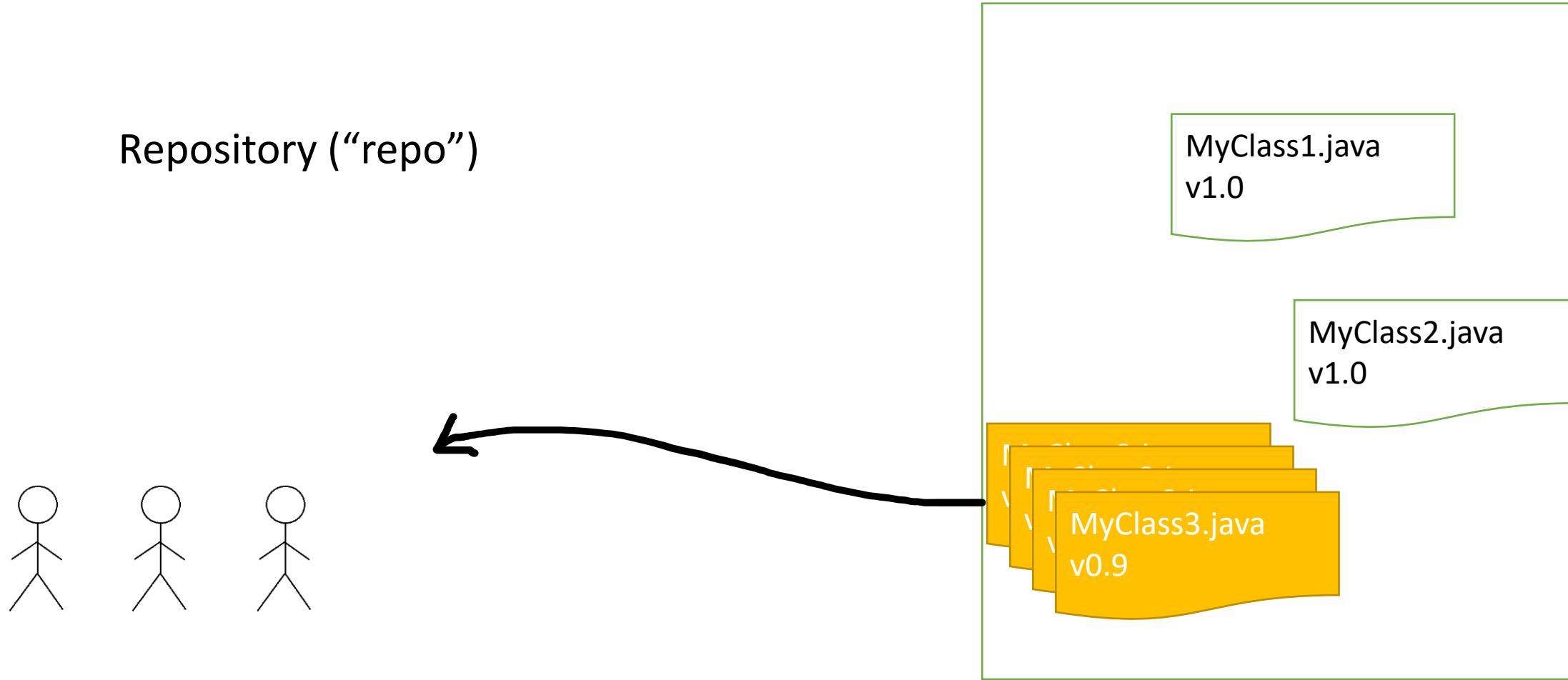
Another category of useful tools are source control systems. They are repositories where the source code for a project is stored. They ensure that only one developer modifies a file at a time, and they also keep track of changes, allowing to revert back in time when changes were bad. There are also some more advanced functions for merging parts that have evolved independently.

Source Control Systems



A check-in/check-out system makes files read-only when someone is modifying them.

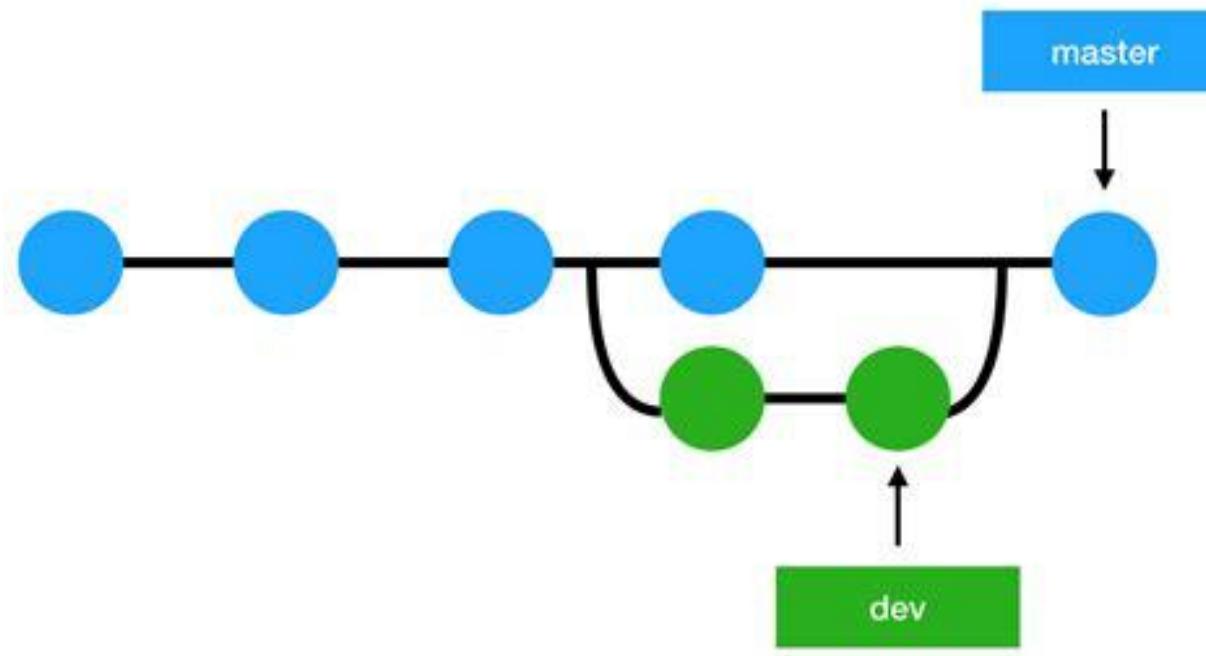
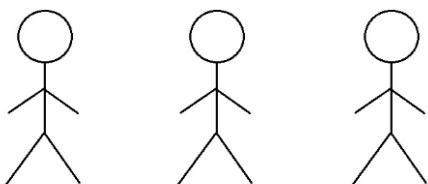
Source Control Systems



Source control systems keep a history with modifications of each file over time enabling rollback.

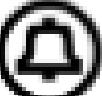
Source Control Systems

Branches



Source control systems have features for merging parts that have evolved independently.

Source Control Systems

- SCCS:  Bell Laboratories
- First one – 1970s was SCCS
- Even earlier systems performing similar functionalities were used by IBM and others at least during the 1960s

Source Code Control System (SCCS) is a per-file revision control system, originally developed by Marc Rochkind at Bell Labs.



There are many systems, the first ones were local to computer, today they can be local or use a web server

Installing a library for Optimization using Evolutionary Computation

Maven, Git, and more

Toolkits you can use to implement genetic algorithms in Java

- ECJ 27: A Java based Evolutionary Computation Research System
<https://cs.gmu.edu/~eclab/projects/ecj/>
- Watchmaker Framework:
<http://watchmaker.uncommons.org/changelog.php>
- JGAP: Java Genetic Algorithm Package: <http://jgap.sourceforge.net/>

Toolkits you can use to implement genetic algorithms in Python

- PyGMO: <https://pagmo.sourceforge.io/pygmo/index.html>
- DEAP: <https://github.com/DEAP/deap>
- Inspyred: <https://aarongarrett.github.io/inspyred/>
- ...

There are others too, this is just a selection

What is ECJ 27?

- A Java based Evolutionary Computation Research Environment
- <https://cs.gmu.edu/~eclab/projects/ecj/>
 - GUI with charting
 - Platform-independent checkpointing and logging
 - Hierarchical parameter files
 - Multithreading
 - Abstractions for implementing a variety of EC features
 - Optional builds with Apache Maven
- Many different algorithms are included
- Easy to define a problem, an algorithm and a search space
- Supported in the research community
- Good documentation

Saw in lecture

EC Features

- Asynchronous island models over TCP/IP
- Master/Slave evaluation over multiple processors, with support for generational, asynchronous steady-state, and coevolutionary distribution
- Genetic Algorithms/Programming style Steady State and Generational evolution, with or without Elitism
- Evolutionary-Strategies style (μ, λ) and ($\mu + \lambda$) evolution
- Dedicated package for efficient single-state (hill-climbing, simulated annealing, etc.) methods
- CMA-ES, AMaLGaM IDEA, PBIL, DOvs EDAs
- Ant System, Ant Colony System, and GRASP
- Very flexible breeding architecture
- Many selection operators
- Multiple subpopulations and species
- Inter-subpopulation exchanges
- Reading populations from files
- Single- and Multi-population coevolution
- NSGA-II, NSGA-III, and SPEA2 multiobjective optimization
- Particle Swarm Optimization
- Differential Evolution
- Spatially embedded evolutionary algorithms
- Hooks for other multiobjective optimization methods
- Meta-Evolution
- Packages for parsimony pressure

Getting it up and running with Maven

- First we will install Maven (this is technically optional in order to get it up and running but Maven is widely used so we learn how to use it and later it will make the package management and installation easier and less prone to errors and having to make manual adjustments)



The screenshot shows a web browser displaying the Apache Maven Project download page at maven.apache.org/download.cgi. The page features the Apache logo and the text "Apache Maven Project" with the URL <http://maven.apache.org/>. On the right, there is a large "Maven" logo with a feather icon. Below the logo, there are links for "Download" and "Get Sources", and a note about the last publication date: "Last Published: 2021-03-27". The main content area is titled "Downloading Apache Maven 3.6.3" and states that it is the latest release. It includes a note about the selected download mirror (<https://mirror.novg.net/apache/>) and instructions for selecting a different mirror if needed. A sidebar on the left contains links for "Welcome", "License", "ABOUT MAVEN", "What is Maven?", "Features", and "Download". The "Download" link is highlighted with a blue background.

→ Download

Apache / Maven / Download Apache Maven

Download Apache Maven 3.6.3

Apache Maven 3.6.3 is the latest release and recommended version for all users.

The currently selected download mirror is <https://mirror.novg.net/apache/>. If you encounter a problem with this mirror please select another mirror. If all mirrors are failing, there are backup mirrors (at the end of the mirrors list) that should be available. You may also consult the [complete list of mirrors](#).

Other mirrors: Change

POM = Project Object Model

- Maven uses the concept of a POM – it is an XML file that contains information about a project and configuration details that are required to build the project
 - EG the build directory
 - The test directory
- When we install Maven it has a default POM which is always used, you can override or even update this default
- A simple POM.XML:
 - project root
 - modelVersion - should be set to 4.0.0
 - groupId - the id of the project's group
 - artifactId - the id of the artifact (product)
 - version - the version of the artifact
- https://maven.apache.org/guides/introduction/introduction-to-the-pom.html#What_is_a_POM

```
1. <project>
2.   <modelVersion>4.0.0</modelVersion>
3.
4.   <groupId>com.mycompany.app</groupId>
5.   <artifactId>my-app</artifactId>
6.   <version>1</version>
7. </project>
```

Install Maven

- It is easy to install Maven
- Just unzip: `unzip apache-maven-3.x.y-bin.zip`
- Open and read the README file
follow the instructions
- Also check the system requirements
- <https://maven.apache.org/install.html>

Installing Maven

- 1) Unpack the archive where you would like to store the binaries, e.g.:

Unix-based operating systems (Linux, Solaris and Mac OS X)

```
tar zxvf apache-maven-3.x.y.tar.gz
```

Windows

```
unzip apache-maven-3.x.y.zip
```

- 2) A directory called "apache-maven-3.x.y" will be created.

- 3) Add the bin directory to your PATH, e.g.:

Unix-based operating systems (Linux, Solaris and Mac OS X)

```
export PATH=/usr/local/apache-maven-3.x.y/bin:$PATH
```

Windows

```
set PATH="c:\program files\apache-maven-3.x.y\bin";%PATH%
```

- 4) Make sure JAVA_HOME is set to the location of your JDK

- 5) Run "mvn --version" to verify that it is correctly installed.

For complete documentation, see <https://maven.apache.org/download.html#Installation>

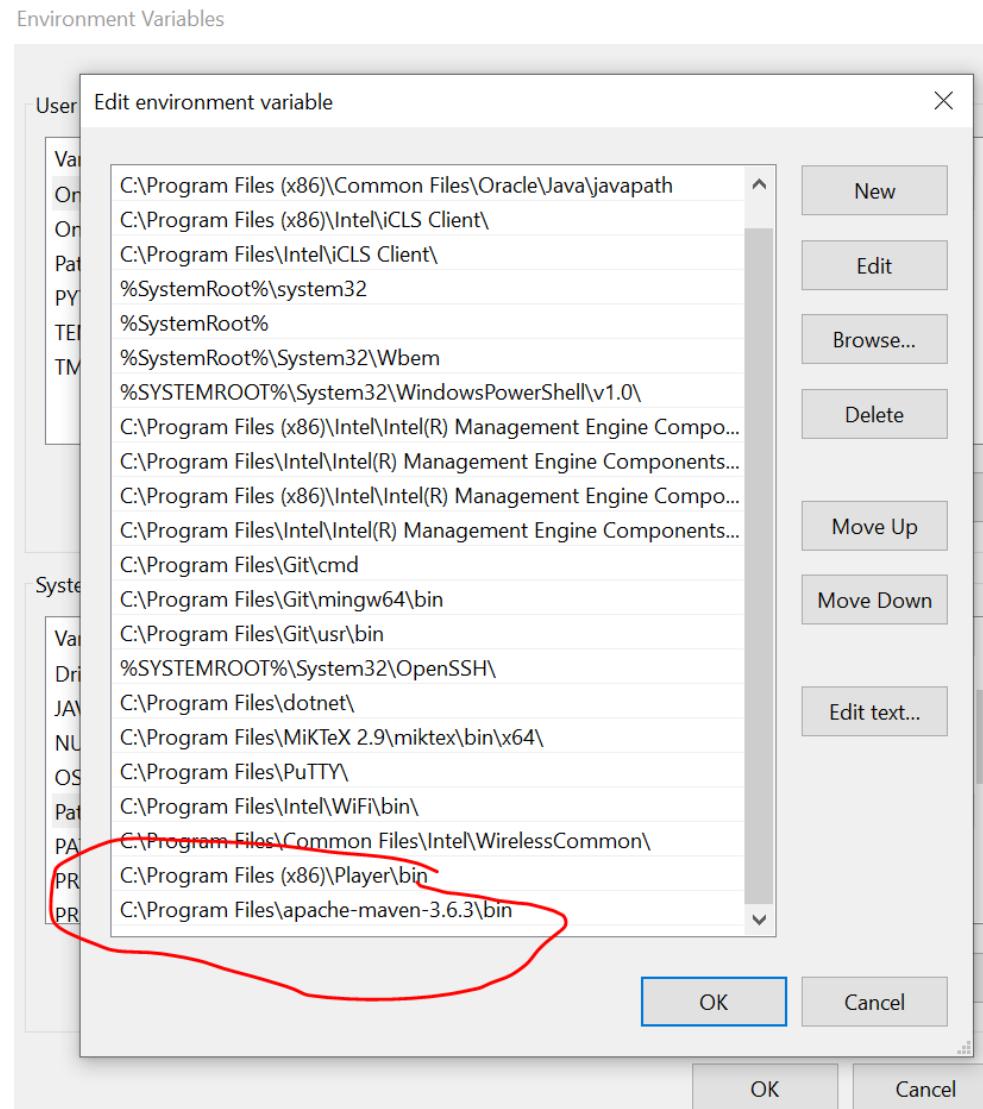
Setting environment variables

In command line:

```
set PATH="c:\program files\
```

Setting environment variables

- Dialog

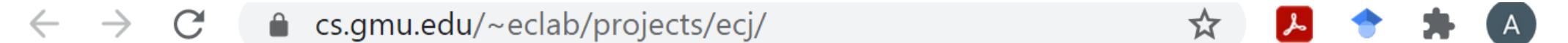


Checking Maven is installed...

```
C:\Users\adamg>mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: c:\Program Files\apache-maven-3.6.3\bin\..
Java version: 13, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-13
Default locale: en_AU, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\adamg>
```

How to setup ECJ27



ECJ 27

A Java-based Evolutionary Computation Research System

By Sean Luke, Eric O. Scott, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Rafal Kicinger, Elena Popovici, Keith Sullivan, Joseph Harrison, Jeff Bassett, Robert Hubley, Ankur Desai, Alexander Chircop, Jack Compton, William Haddon, Stephen Donnelly, Beenish Jamil, Joseph Zelibor, Eric Kangas, Faisal Abidi, Houston Mooers, James O'Beirne, L. Manzoni, Khaled Ahsan Talukder, Sam McKay, James McDermott, Jason Zou, Anson Rutherford, David Freelan, Ermo Wei, Sunil Rajendran, Ananya Dhawan, Ben Brumbac, Javier Hilty, and Anowarul Kabir.

ECJ is a research EC system written in Java. It was designed to be highly flexible, with nearly all classes (and all of their settings) dynamically determined at runtime by a user-provided parameter file. All structures in the system are arranged to be easily modifiable. Even so, the system was designed with an eye toward efficiency.

ECJ is developed at George Mason University's [ECLab](#) Evolutionary Computation Laboratory. The software has nothing to do with its initials' [namesake](#), *Evolutionary Computation Journal*. ECJ's sister project is [MASON](#), a multi-agent simulation system which dovetails with ECJ nicely.

Download ECJ version 27 as [ecj27.tar.gz](#) or (bigger) [ecj27.zip](#).

Download ECJ's support libraries as [libraries.tar.gz](#) or [libraries.zip](#). The support libraries provide four facilities:

- ECJ's **CMA-ES** code relies on the matrix library [EJML](#).
- ECJ's **distributed evaluation** and **island model** facilities have optional compressed socket options for more efficiency. Java's standard compression routines are broken for sockets (they don't support "partial flush") and so we rely on the [JZlib](#) library to do it.
- **ECJ's GUI** relies on the [JFreeChart](#) and [iText](#) libraries, specifically the jfreechart.jar, jcommon.jar, and iText.jar files. Use the ECJ versions, not the versions on those websites. If you don't wish to use the GUI and thus install JFreeChart and iText, you can just get rid of the ec/display directory and don't call make gui, and ECJ will compile and run from the command line fine. Per license agreement, we provide source to these libraries [here](#).
- ECJ's **Push facility** relies on a modified version of the [Psh](#) interpreter. Be sure to use the ECJ version.

New! [The ECJ Owner's Manual](#) is now available. Hundreds of pages of mind-numbing tedium detailing every last feature. Enjoy!

The **full online documentation** of the system, including tutorials, is also [available](#).

Repository: ECJ's repository is at Github, under the project name [ecj](#).

Contrib packages: (also out on CVS in the contrib directory)

Previous Releases

- Version 26: [ecj25.tar.gz](#)
- Version 25: [ecj25.tar.gz](#)
- Version 24: [ecj24.tar.gz](#)
- Version 23: [ecj23.tar.gz](#)
- Version 22: [ecj22.tar.gz](#)
- Version 21: [ecj21.tar.gz](#)
- Version 20: [ecj20.tar.gz](#)
- Version 19: [ecj19.tar.gz](#)
- Version 18: [ecj18.tar.gz](#)
- Version 17: [ecj17.tar.gz](#)
- Version 16: [ecj16.tar.gz](#)
- Version 15: [ecj15.tar.gz](#)
- Version 14: [ecj14.tar.gz](#)
- Version 13: [ecj13.ec.tar.gz](#)
- Version 12: [ecj12.ec.tar.gz](#)
- Version 11: [ecj11.ec.tar.gz](#)
- Version 10: [ecj10.ec.tar.gz](#)
- Version 9: [ecj9.ec.tar.gz](#)
- Version 8: [ecj8.ec.tar.gz](#)
- Version 7: [ecj7.ec.tar.gz](#)
- Version 6: [ecj6.ec.tar.gz](#)
- Version 5: [ecj5.ec.tar.gz](#)
- Version 4: [ecj4.ec.tar.gz](#)
- Version 3: [ecj3.ec.tar.gz](#)

How to setup ECJ27

- We read the README files after we download ECJ27 and decide use Maven to build it (“install”)

```
C:\Users\adamg\Desktop\Java Systems 2021\ecj27\ecj>pwd  
/c/Users/adamg/Desktop/Java Systems 2021/ecj27/ecj/ecj
```

- After you decompress the downloaded ecj file (with unzip or tar –xvf) use a command prompt to navigate to the directory shown (list the contents of a directory in windows with dir in unix can use “ls” and “pwd” to list the files or print the current directory)

1.3.1 The ec Directory, the CLASSPATH, and jar files

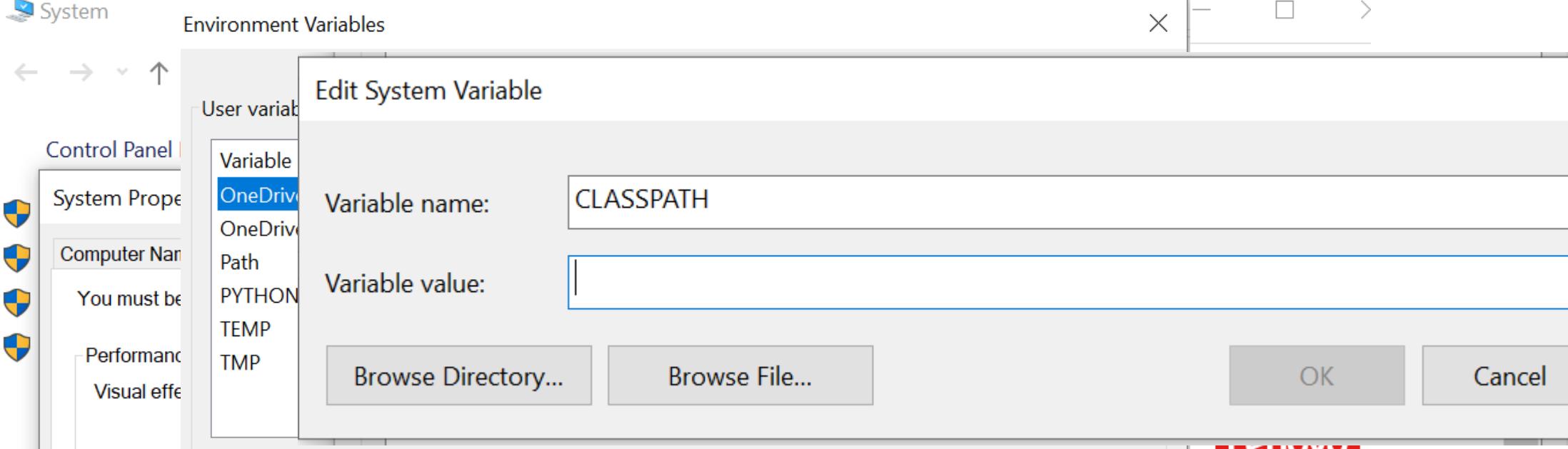
The `ec` (`ecj/src/main/java/ec/`) directory is ECJ's top-level package. Every subdirectory is a subpackage, and most of them are headed by helpful README files which describe the contents of the directory. Most packages contain not only Java files and class files but also parameter files and occasional data files: ECJ was designed originally for the class files to be compiled and stored right alongside the Java files in these directories, though it can be used with the separate-build-area approach taken by IDEs like Eclipse.

Because `ec` is the top-level package, you can compile ECJ, more or less, by just sticking its parent directory (the `ecj/target/` directory), in your CLASSPATH. You will also need to add certain **jar files** in order to compile ECJ's distributed evaluation and island model facilities, and its GUI. You can get these jar files from the ECJ main website (<http://cs.gmu.edu/~eclab/projects/ecj/>). Note that none of these libraries is required. For example, if the libraries for the distributed evaluator and island model are missing, ECJ will compile but will complain if you try to run those packages with compression turned on (a feature of the packages). The GUI library is optional to ECJ, so if you don't install its libraries, you can still compile ECJ by just deleting the `ec/display` directory.

Adding the required libraries to the Classpath

- You can also edit system environment variables on windows in the configuration and settings:
- Windows Computer->Properties->Advanced Dialog
- Also called “Advanced System Settings”

| Name |
|-------------------|
| EJML-core-0.27 |
| EJML-dense64-0.27 |
| EJML-simple-0.27 |
| iText-1.2 |
| jcommon-1.0.16 |
| jfreechart-1.0.13 |
| jzlib-1.0.7 |
| pshecj |
| README |



| | | New... | Edit... | Delete |
|----------------------|--|--------|---------|--------|
| System variables | | | | |
| Variable | Value | | | |
| CLASSPATH | C:\Users\adamg\Desktop\Java Systems 2021\libraries | | | |
| ComSpec | C:\WINDOWS\system32\cmd.exe | | | |
| configsetroot | C:\WINDOWS\ConfigSetRoot | | | |
| DriverData | C:\Windows\System32\Drivers\DriverData | | | |
| NUMBER_OF_PROCESSORS | 4 | | | |
| OS | Windows_NT | | | |
| Path | C:\Program Files\Java\jdk-13\bin;C:\Program Files (x86)\Common... .COM;.FXF;.BAT;.CMD;.VBS;.VBF;.JS;.JSF;.WSF;.WSH;.MSC | | | |
| Variable | Value | New... | Edit... | Delete |

Change settings

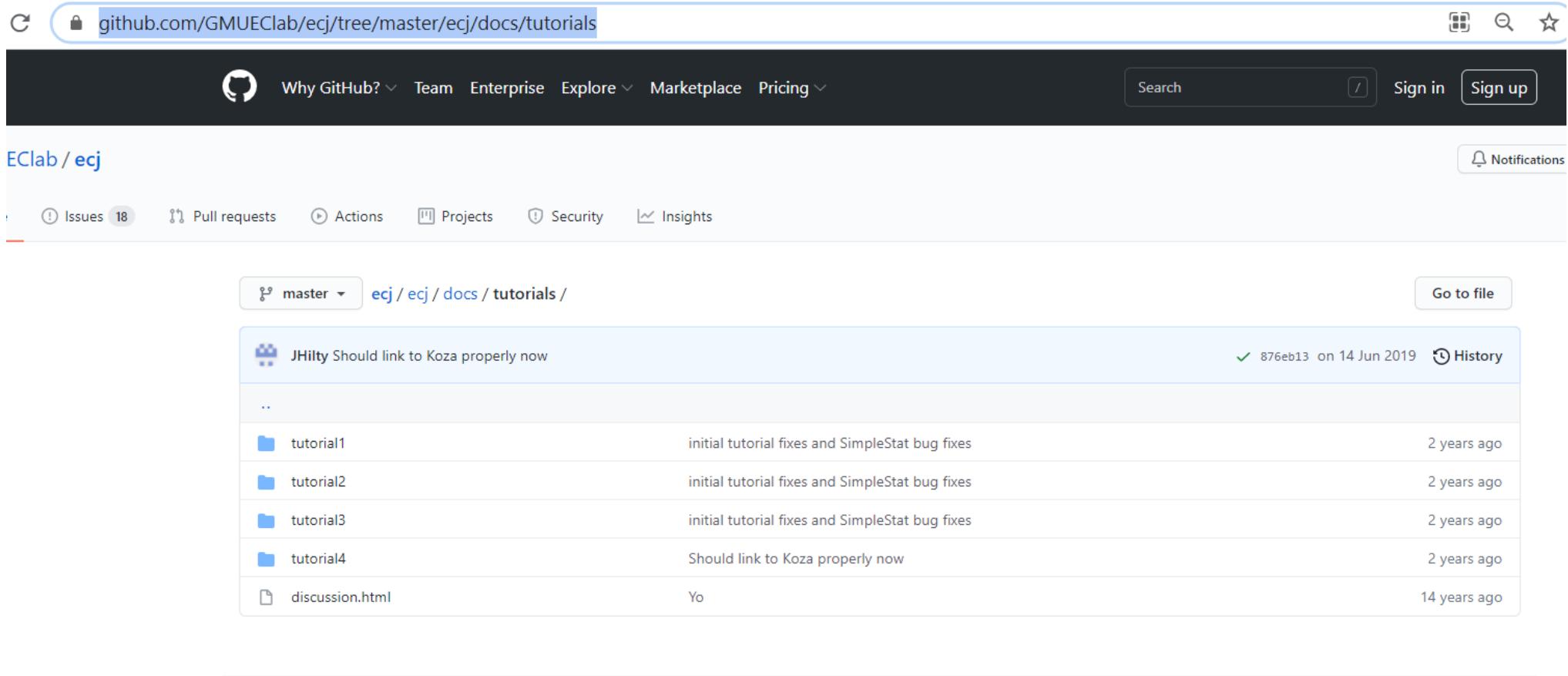
Installing ecj

- Review the readme file: ..\ecj27\ecj\ecj\README.md

```
C:\Users\adamg\Desktop\Java Systems 2021\ecj27\ecj>cd ecj  
C:\Users\adamg\Desktop\Java Systems 2021\ecj27\ecj\ecj>ls  
CHANGES LICENSE Makefile README.md docs front.stat local-repo out.stat pom.xml src  
C:\Users\adamg\Desktop\Java Systems 2021\ecj27\ecj\ecj>
```

- mvn clean package
 - ... Will download various files that ecj depends on...
 - ...Then compile ecj to “target/ecj-xx.jar”
 - ...Notice it also runs tests

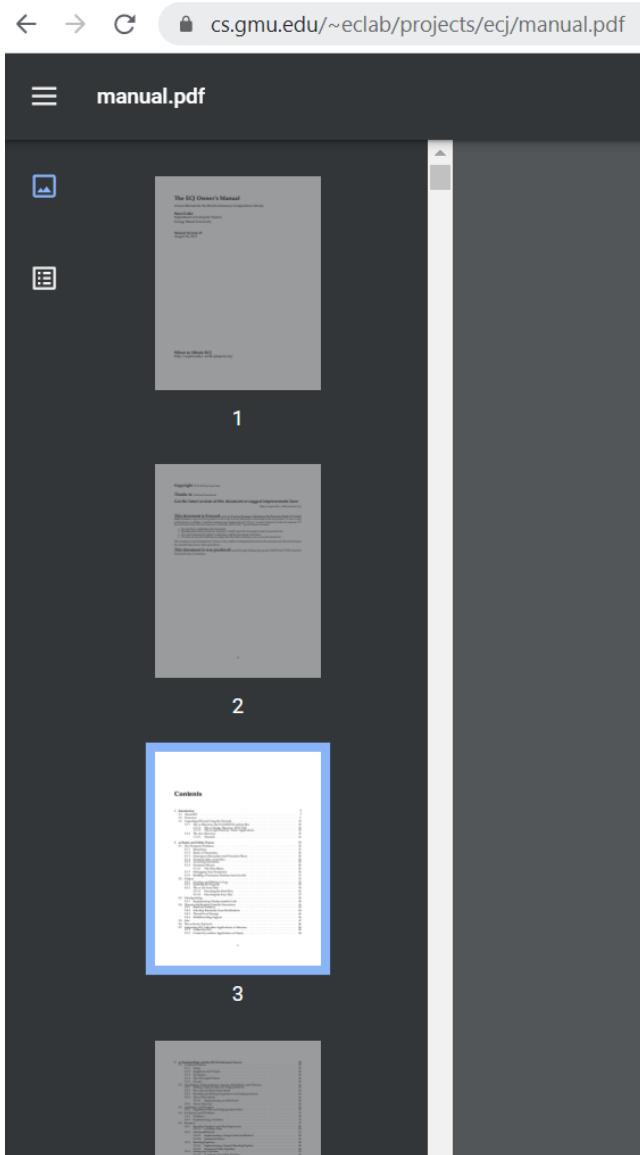
Tutorials



A screenshot of a GitHub repository page for the 'ecj' branch of the 'EClab' repository. The URL in the address bar is `github.com/GMUEClab/ecj/tree/master/ecj/docs/tutorials`. The page shows a list of files in the 'tutorials' directory. The files listed are:

- `tutorial1`: initial tutorial fixes and SimpleStat bug fixes (2 years ago)
- `tutorial2`: initial tutorial fixes and SimpleStat bug fixes (2 years ago)
- `tutorial3`: initial tutorial fixes and SimpleStat bug fixes (2 years ago)
- `tutorial4`: Should link to Koza properly now (2 years ago)
- `discussion.html`: Yo (14 years ago)

Manual



The screenshot shows a PDF viewer interface with the following details:

- Title Bar:** The URL bar displays "cs.gmu.edu/~eclab/projects/ecj/manual.pdf".
- Toolbar:** Includes standard PDF navigation icons (back, forward, search, etc.) and a zoom control set at 100%.
- Page Number:** The page number "3 / 317" is visible in the top center.
- Content Area:** The main area shows the first three pages of the ECJ manual.
 - Page 1:** The title page of the ECJ Owner's Manual.
 - Page 2:** The copyright page, which includes the ECJ license information.
 - Page 3:** The contents page, which lists all chapters and their page numbers.
- Table of Contents:** A detailed table of contents is displayed on the right side of the screen, listing all chapters and their corresponding page numbers.

| Chapter | Page Number |
|--|-------------|
| 1 Introduction | 7 |
| 1.1 About ECJ | 7 |
| 1.2 Overview | 9 |
| 1.3 Unpacking ECJ and Using the Tutorials | 14 |
| 1.3.1 The ec Directory, the CLASSPATH, and jar files | 15 |
| 1.3.1.1 The ec/display Directory: ECJ's GUI | 15 |
| 1.3.1.2 The ec/app Directory: Demo Applications | 15 |
| 1.3.2 The docs Directory | 15 |
| 1.3.2.1 Tutorials | 16 |
| 2 ec.Evolve and Utility Classes | 17 |
| 2.1 The Parameter Database | 18 |
| 2.1.1 Inheritance | 19 |
| 2.1.2 Kinds of Parameters | 20 |
| 2.1.3 Namespace Hierarchies and Parameter Bases | 22 |
| 2.1.4 Parameter Files in Jar Files | 24 |
| 2.1.5 Accessing Parameters | 25 |
| 2.1.6 Parameter Macros | 28 |
| 2.1.6.1 The Alias Macro | 28 |
| 2.1.7 Debugging Your Parameters | 29 |
| 2.1.8 Building a Parameter Database from Scratch | 31 |
| 2.2 Output | 33 |
| 2.2.1 Creating and Writing to Logs | 33 |
| 2.2.2 Quieting the Program | 35 |
| 2.2.3 The ec.util.Code Class | 35 |
| 2.2.3.1 Decoding the Hard Way | 36 |
| 2.2.3.2 Decoding the Easy Way | 37 |
| 2.3 Checkpointing | 38 |
| 2.3.1 Implementing Checkpointable Code | 40 |
| 2.4 Threads and Random Number Generation | 41 |
| 2.4.1 Random Numbers | 41 |
| 2.4.2 Selecting Randomly from Distributions | 44 |
| 2.4.3 Thread-Local Storage | 46 |
| 2.4.4 Multithreading Support | 46 |

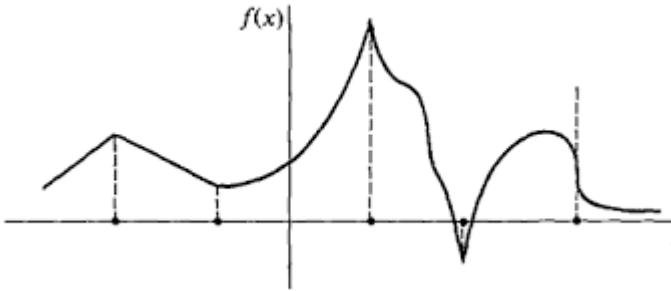
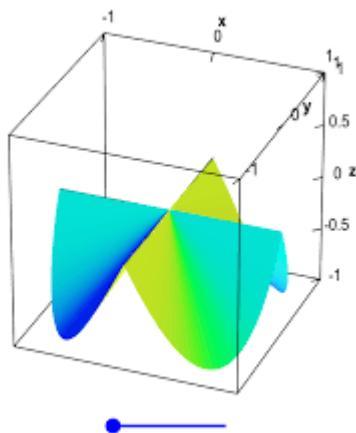
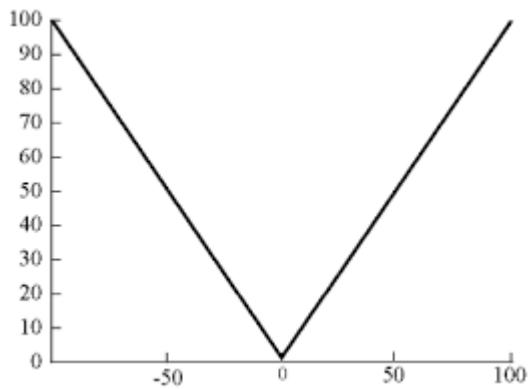
69

Homework

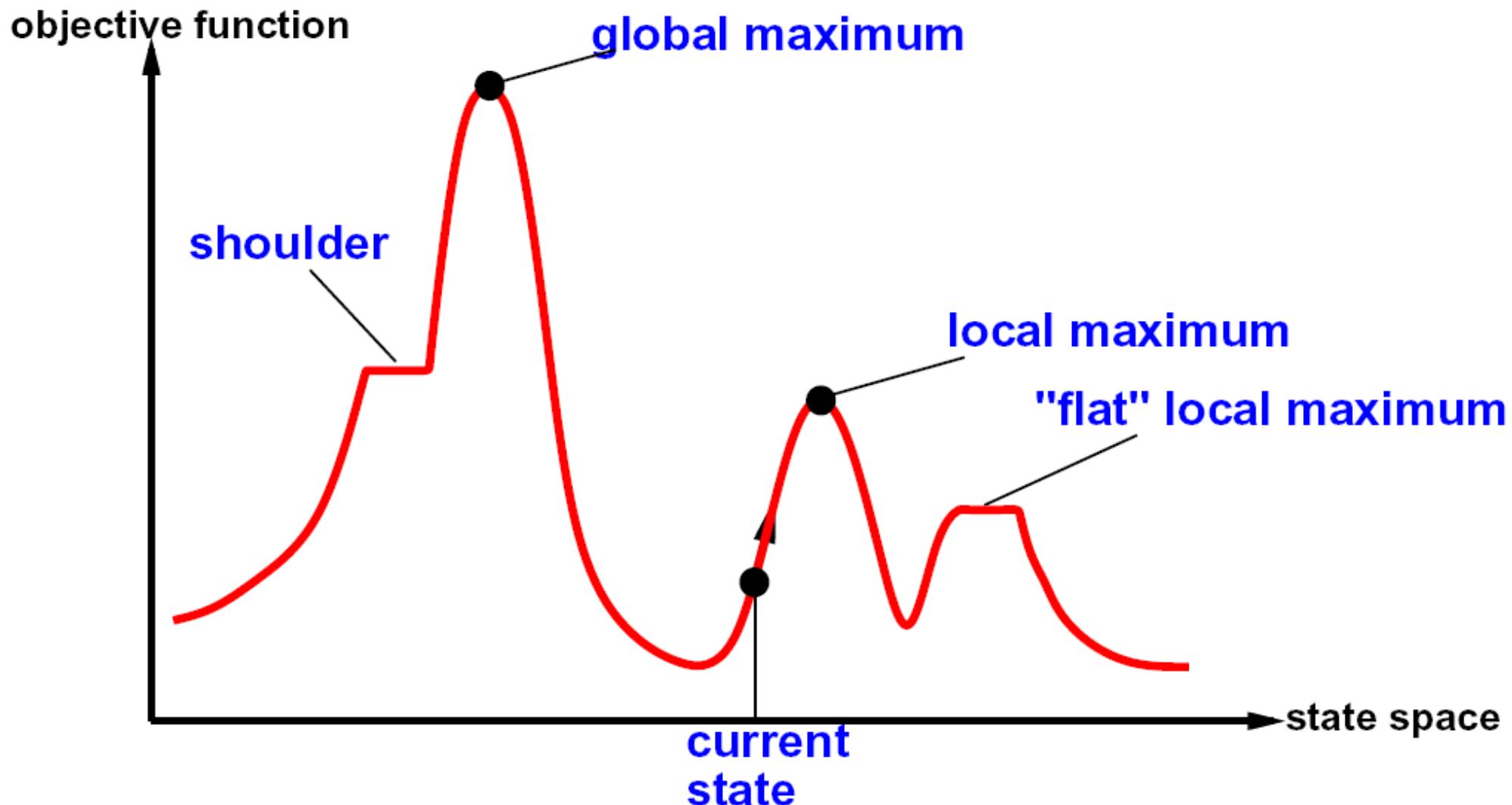
- Please install ECJ27 on your laptop and run the tutorials on the previous slide (use Maven to build and install dependencies and Github to access the tutorials) nb: the tutorials are from the github site for ecj 27
- Please try to use the mu-lambda and mu+lambda algorithms presented in the Lecture 5 (when we looked at optimisation methods to use in the project)

Problem Types for Solvers

- Non linear
- Non differentiable
- Non convex
- Large and complex



Searching

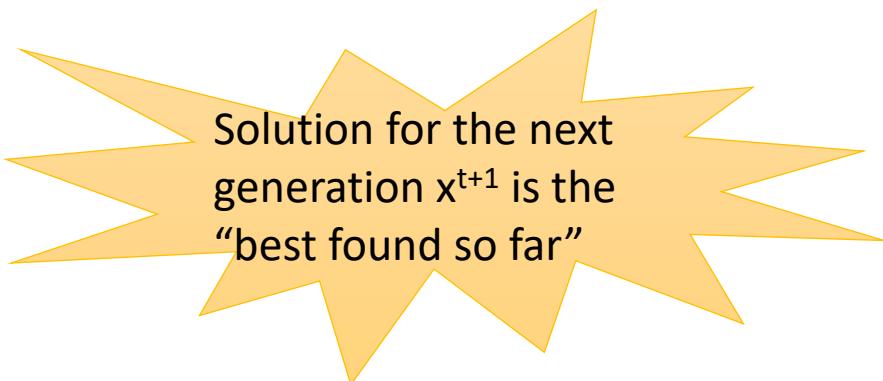


Example – algorithm

1. Set $t = 0$
2. Create initial point at random: $\mathbf{x}^t = \langle x_1^t, \dots, x_n^t \rangle$
3. REPEAT UNTIL (*TERMIN.COND* satisfied) DO
4. Draw \mathbf{z}_i from a normal distribution for all $i = 1, \dots, n$
5. $\mathbf{y}^t = \mathbf{x}^t + \mathbf{z}_i$
6. IF $\text{fitness}(\mathbf{x}^t) < \text{fitness}(\mathbf{y}^t)$ THEN $\mathbf{x}^{t+1} = \mathbf{x}^t$ ELSE $\mathbf{x}^{t+1} = \mathbf{y}^t$
7. Set $t = t+1$
8. OD



The operator to get a new solution is to add z to the previous solution



Solution for the next generation \mathbf{x}^{t+1} is the “best found so far”



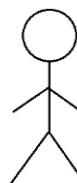
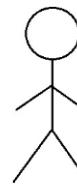
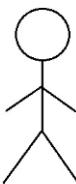
Testing

Testing

A very important phase in the life of a developer is testing, which is both kind of boring and difficult to do properly. Testing is a task that has to be done repeatedly, because very often a change (new feature, bug fix) breaks something that used to work. For big projects, you have "test suites" that just run the software through a lot of controls and checks that everyone of them is passed.

Testing

Program



Testing is more difficult when several developers are working on the same project.

The fastest developer cannot wait on the slowest one to test the code!

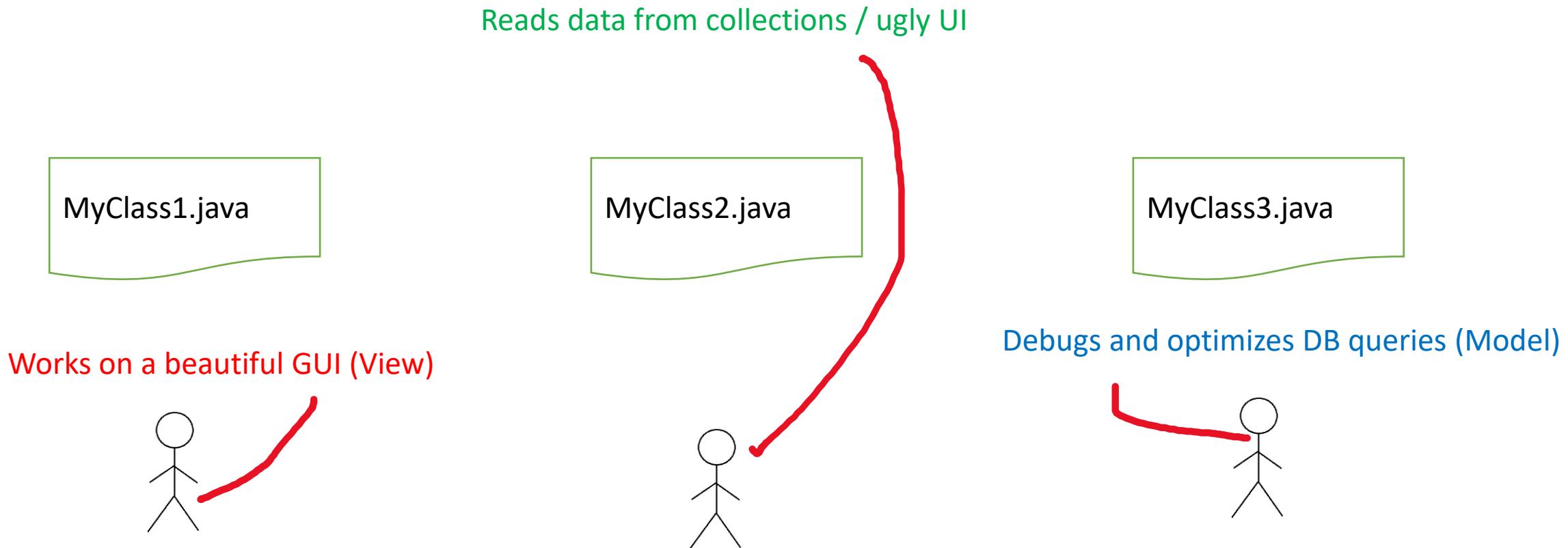
You usually want to test your code as early as possible, even if you are using objects and methods currently being developed by someone else (remember that object-oriented programming is mostly objects exchanging messages by calling methods)

Write dummy classes and methods for testing



The solution is to write very simple objects and methods just for simulating the real thing (that is not ready or in another component). EG instead of actually getting data from a DB you can return the same data from a small hard-coded or configured collection. Instead of getting a real message from a remote server you can make a fake one. The word "mock" meaning imitation or fake is often used, it can be found in the names of products that are useful for generating (with reflection) code that is used in testing.

Testing



This allows developers to test their code without having to wait for others – or without having written all their methods.

Testing: Dependency Injection

An important idea for testing is that when an object depends on another object from a different class, it should not create the other object, but should get a reference to it. The dependency is "injected" (passed). This makes testing far easier, because you don't have to worry about what the constructor should look like and what arguments it takes. Dependency injection is central to some development frameworks such as Spring and is considered a good development practice.

Aspects of Testing

Testing covers many fields – including the behaviour when something that wasn't expected happens.

- Expected outcome/ result?
- New change doesn't break something? (regression)
 - Potentially a very large set of small tests that are run to make sure when a new piece of code is added to a large system it does not cause it to regress to be less useful (that is to reduce the current functions it is expected to have)
- What the user wanted?
 - User Acceptance Test
- Correct performance
 - EG load testing

Testing Tools

Also see TestNG



Some tests are usually carried out by support teams, not by developers themselves. For the testing part that directly regards developers, some tools exist that are based on annotations.

JUnit

```
import static org.junit.Assert.*;  
import org.junit.*
```

```
class MyClass
```

```
public int method1{  
}  
  
public int method2{  
}
```

```
@test  
public int method1{  
}  
  
@test  
public int method2{  
}
```

The idea is to mirror a class with a test class that checks, in a test method, a method from the original class.

Test Methods

A test method, annotated as such, uses an assertxxx() function to compare the result of a method to test to an expected result.

```
@Test  
public void testmethod1 {  
    // Create object,  
    // initialize parameters ...  
    assertEquals(expected_result, obj.method1(...));  
}
```

There are many assertxxx methods, that can optionally take a message as parameter.

- assertEquals("message",A,B);
- assertTrue(A);
- assertFalse(A);
- assertNotNull(A);
- ...

Junit Annotations

@Test

@Before

@After

@Test(expected = Exception.class)

@Test(timeout = 100)

Annotations allow to define "before" and "after" operations, and even to check that we are getting the proper exception.

Testing Setup

- "Test suites"
- **Ideally test only one class**
- **As many test methods as you need**

Normally you are supposed to test one class at once, and you can have multiple test methods to test different aspects (there is NOT a one-to-one correspondance between methods being tested and test methods).

Tests can be run from multiple environments.

- Running JUnit Tests (Tests can be run from multiple environments).
 - IDE
 - Build tool
 - Command line: \$ java org.junit.runner.JUnitCore TestClass1 [...other test classes...]
- You can use Maven to automatically run a test suite when you build your application (this is a common practice)

maven

Deployment

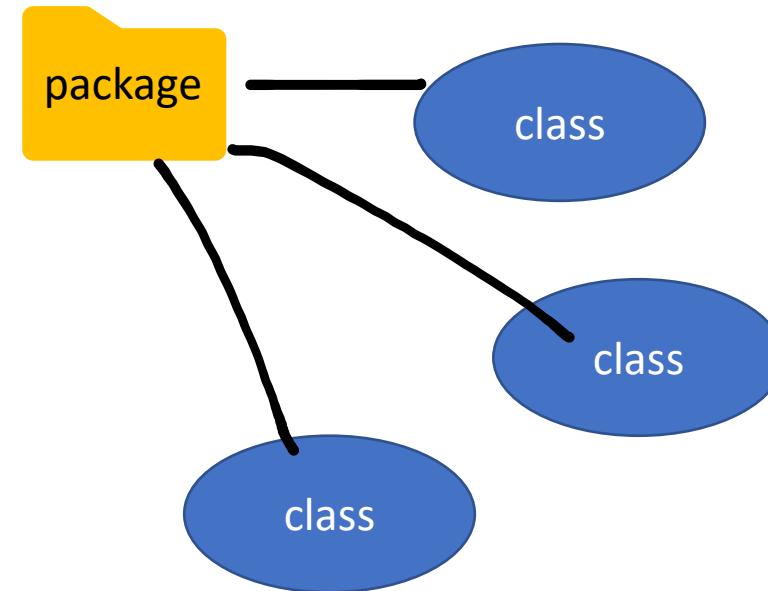
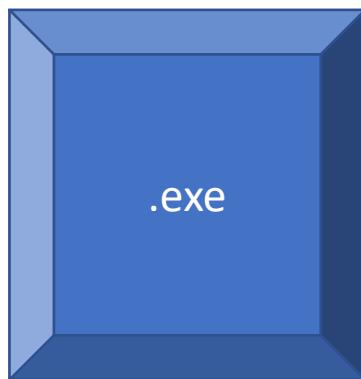


Deployment: *Distributing the program*

The last aspect isn't the least important. How are you going to distribute your program? Cases when the program is a single .class are very rare. Usually you need quite a number of files to successfully run a program.

Deployment: Executable Files

Compared to a standard .exe file in Windows, Java is a mess. You may need several .class files, as well as one or several packages, to successfully run your program.



Deployment: Jar Files

- When you need to send files by email you sometimes zip them into an archive.
 - You do the same thing in java with a .jar file
 - The JVM knows how to read and execute a .jar file without having to unzip it first.
-
- Either a complementary library (e.g. JDBC drivers)
`java -cp somefile.jar`
 - Or the main program
`java -jar myprogram.jar`

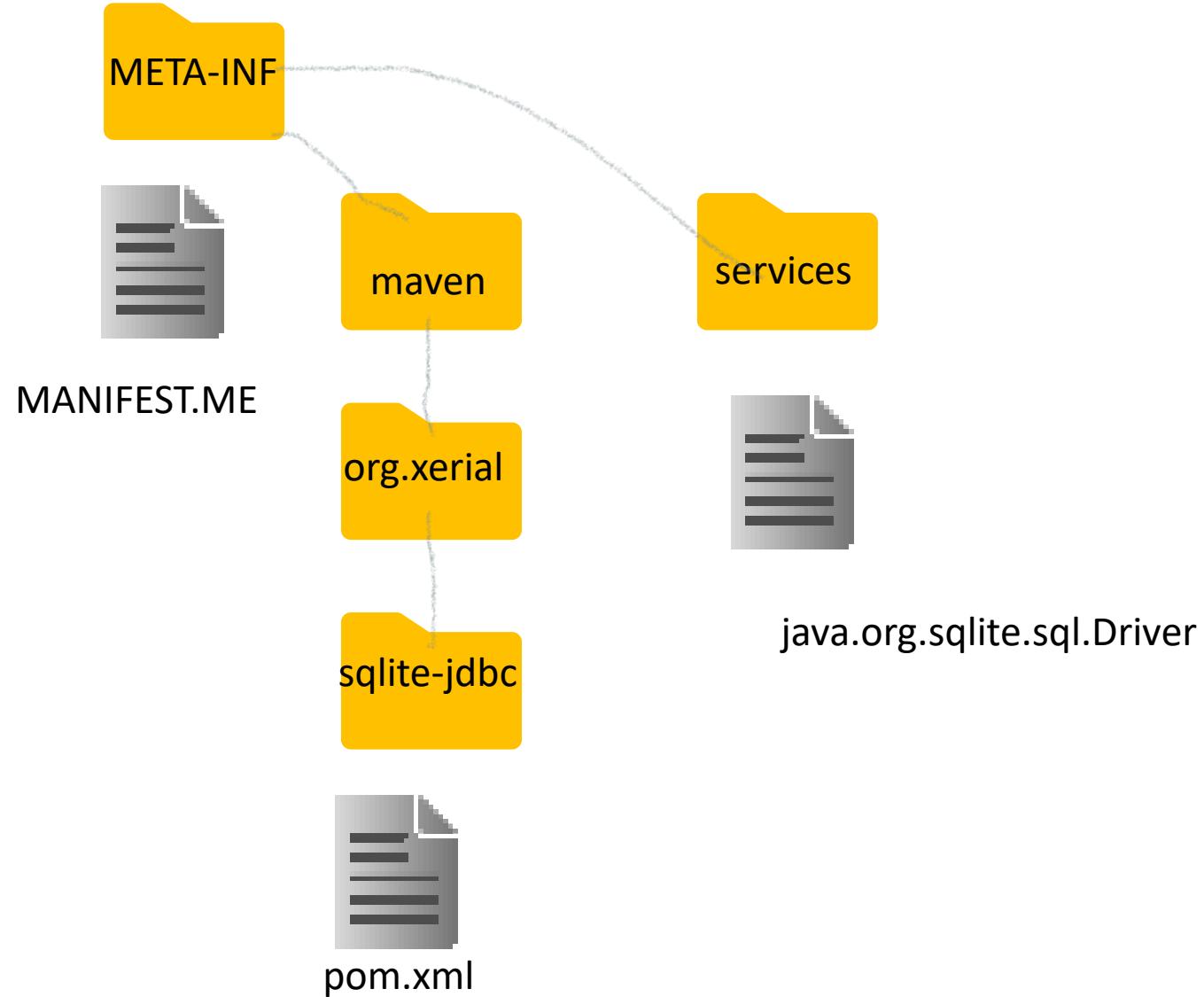
JAR files: Java Archive

"jar" means "java archive", it's inspired by "tar" (tape archive), an old Unix command. It's also a pun, as a jar is usually a glass or earthenware container with a wide opening.



Technically, a .jar file is a compressed (zip) file.

In practice, it IS a zip file, and you can apply unzip to a .jar. Which we will do for the SQLite driver as an exercise.



This is what you find, with obvious traces of Maven.
The manifest describes the contents of the .jar

Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven Bundle Plugin
Built-By: leo
Build-Jdk: 1.8.0_192
Bnd-LastModified: 1577134259111
Bundle-Description: SQLite JDBC library
Bundle-License: <http://www.apache.org/licenses/LICENSE-2.0.txt>
Bundle-ManifestVersion: 2
Bundle-Name: SQLite JDBC
Bundle-SymbolicName: org.xerial.sqlite-jdbc;singleton:=true
Bundle-Version: 3.30.1
Export-Package: org.sqlite;version="3.30.1.SNAPSHOT";uses:="javax.sql, org.sqlite.core,org.sqlite.date",org.sqlite.core;version="3.30.1.SNAPSHOT";uses:="org.sqlite,org.sqlite.jdbc4",org.sqlite.util;version="3.30.1.SNAPSHOT",org.sqlite.date;version="3.30.1.SNAPSHOT",org.sqlite.javax;version="3.30.1.SNAPSHOT";uses:="javax.sql,org.sqlite,org.sqlite.jdbc4",org.sqlite.jdbc4;version="3.30.1.SNAPSHOT";uses:="javax.sql,org.sqlite,org.sqlite.core,org.sqlite.jdbc3",org.sqlite.idbc3;version="3.30.1.SNAPSHOT";uses:="org.sqlite,org.sqlite.core"
Import-Package: javax.sql;resolution:=optional
Originally-Created-By: Apache Maven Bundle Plugin
Tool: Bnd-2.1.0.20130426-122213

This is the content of the Manifest file.

Jar Files

References to files in a .jar are supposed to be in the .jar, unless they are prefixed by “file:”

References to files:

- file:path
- Operating System

look in the CLASSPATH variable

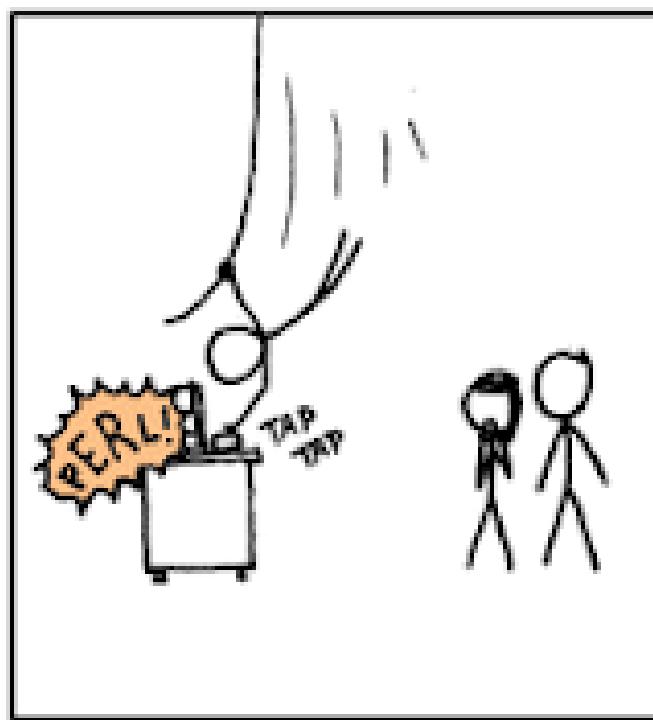
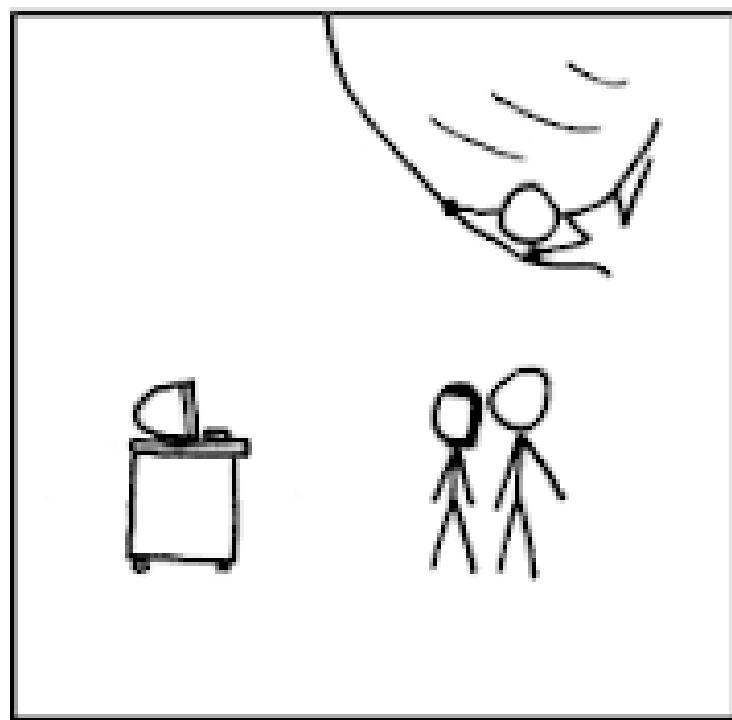
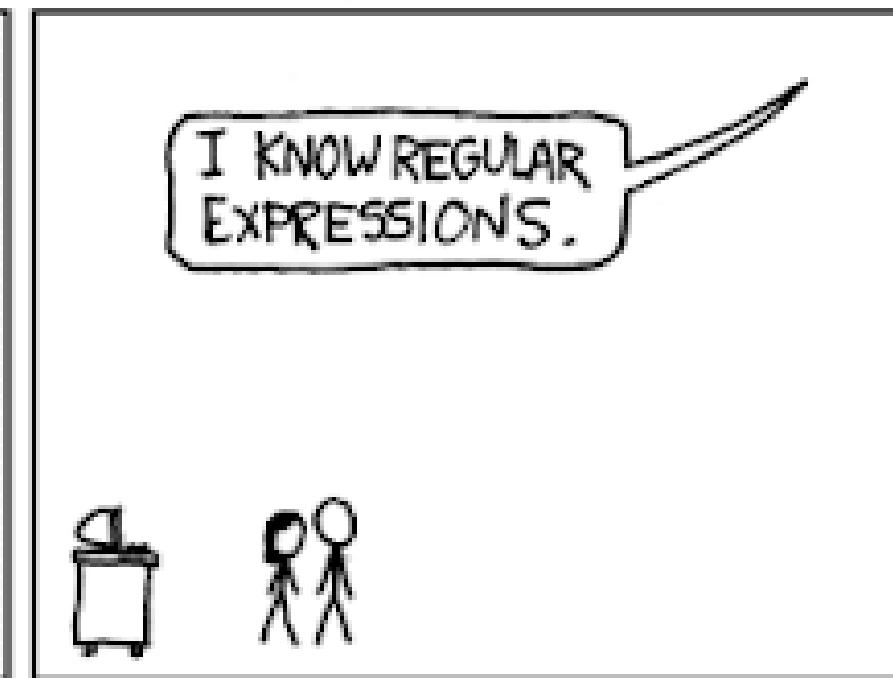
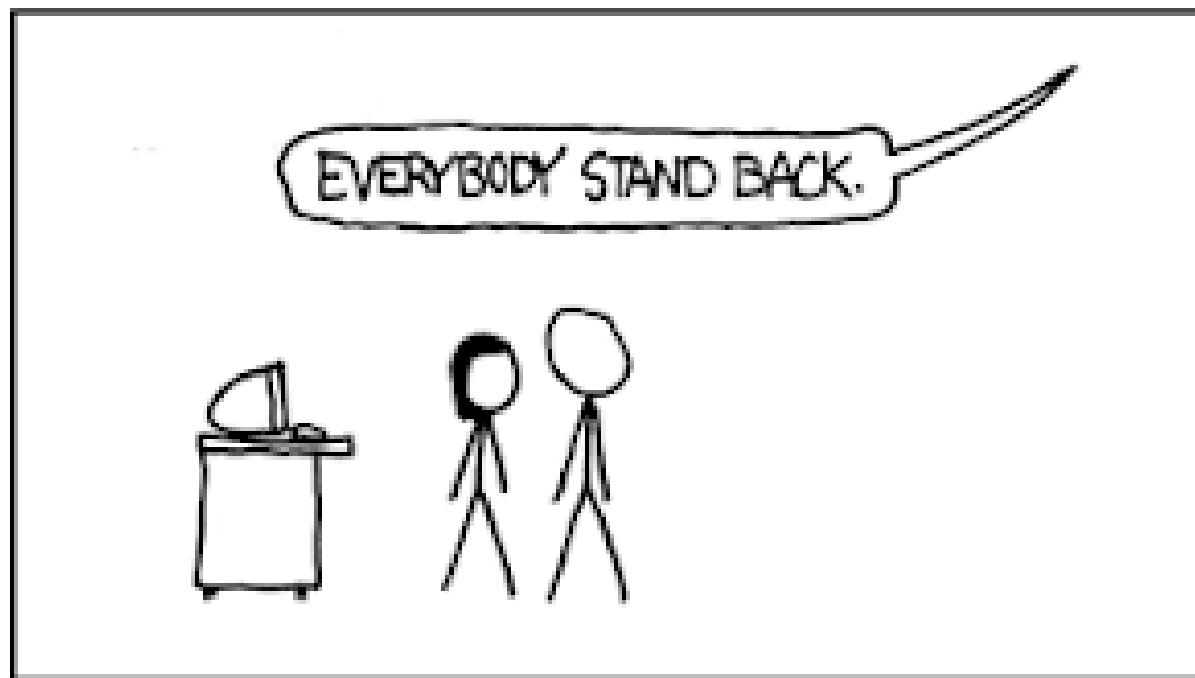
```
this.getClass()  
    .getClassLoader()  
    .getResource("images/image.png")  
    images must be in CLASSPATH ↴
```

Regex

NOW, THIS PART
OF THE REGEX

(?::(\d))?(?:\/([^?#]*))?(?:\?([^





New Problem Type: String Pattern Matching

- Problem description
 - **Input:** a **pattern** and a **text**
 - **Output:** Occurrences of the pattern in the text
- Applications:
 - Searching in a document
 - Finding web pages relevant to queries
 - Searching for patterns in DNA sequences

Examples of String Matching

Input:

Main String: "CODESDOPE", Pattern: "CODE"

Output:

Pattern found at location: 0

Input:

Main String: "ABAAABAAAAAABBAAAABA", Pattern: "AAAB"

Output:

Pattern found at location: 2

Pattern found at location: 8

Pattern found at location: 14

New Problem Type: Regular Expressions and String Pattern Matching

In computing, a regular expression, also referred to as "regex" or "regexp", provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.

Regular Expressions

- Regular expression (also called regex or RE) is useful for describing search patterns for matching text.
- A Regex is just a sequence of some characters that defines a search pattern.
- Regex are applied widely for parsing, filtering, validating, and extracting meaningful information from large text, such as logs and output generated from other programs.

Some Background

- The term *regular expression* comes from *mathematics and computer science theory*, where it reflects a trait of mathematical expressions called *regularity*. Such an expression can be implemented in software using a **deterministic finite automaton (DFA)**. A DFA is a finite state machine that doesn't use backtracking.
- The text patterns used by the earliest *grep tools* were *regular expressions in the mathematical sense*. Though the name has stuck, modern-day Perl-style regular expressions are not regular expressions at all in the mathematical sense.
- They're implemented with a **nondeterministic finite automaton (NFA)**. All a practical programmer needs to remember from this note is that some **ivory tower computer scientists get upset about their well-defined terminology being overloaded with technology that's far more useful in the real world**.

GREP

file

```
[As-MacBook-Pro:Desktop ag$ cat demo.txt
THIS LINE IS THE 1ST UPPER CASE LINE IN THIS FILE.
this line is the 1st lower case line in this file.
This Line Has All Its First Character Of The Word With Upper Case.

Two lines above this line is empty.
And this is the last line.
```

Regex: grep returns lines matching the regex pattern

```
[As-MacBook-Pro:Desktop ag$ 
[As-MacBook-Pro:Desktop ag$ 
[As-MacBook-Pro:Desktop ag$ grep "lines.*empty" demo.txt
Two lines above this line is empty.
[As-MacBook-Pro:Desktop ag$ 
[As-MacBook-Pro:Desktop ag$ 
[As-MacBook-Pro:Desktop ag$ grep -h
usage: grep [-abcDEFGHhIiJLlmnOoqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color=when]
          [--context[=num]] [--directories=action] [--label] [--line-buffered]
          [--null] [pattern] [file ...]
As-MacBook-Pro:Desktop ag$
```

Formal Languages

Some definitions...

- An **alphabet** is a set of symbols
- A **string** is a finite sequence of alphabet symbols
- A **formal language** is a set of strings (possibly infinite if repeated) over the same alphabet

Languages

- Σ^* : “sigma star” denotes the set of finite length strings formed from characters in the alphabet Σ
- ϵ : “epsilon” is the empty string and it also belongs to Σ^*

Example: we have $\Sigma = \{0,1\}$ is the alphabet of binary strings with 0's and 1's

- A language is a subset of Σ^*

$L = \{\epsilon, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, \dots\}$

| <i>formal language</i> | <i>in the language</i> | <i>not in the language</i> |
|---|---|------------------------------------|
| <i>second-to-last symbol is a</i> | aa bbbab bbbbbbbbbababab | a aaaba bbbbbbbbbfffff |
| <i>equal numbers of as and bs</i> | ba bbaaba aaaabbbbbbaaaaba | a bbbaa abababababababa |
| <i>palindromes</i> | a aba abaabaabaaba | ab bbbba abababababababab |
| <i>contain the pattern abba</i> | abba abaabbabbababbba bbbbbbbbbabbabbbb | abb bbabaab aaaaaaaaaaaaaaaa |
| <i>number of bs is divisible by 3</i> | bbb baaaaabaaaab bbbabbbaaabaaabababaaa | bb abababab aaaaaaaaaaaaaaab |

Examples of formal languages over a binary alphabet

More Examples of Formal Languages

| | <i>symbols</i> | <i>symbol name</i> | <i>string name</i> |
|---------------------|--|--------------------|--------------------|
| <i>binary</i> | 01 (or ab) | bit | bitstring |
| <i>Roman</i> | abcdefghijklmnoprstuvwxyz ABCDEFGHIJKLMNPQRSTUVWXYZ | letter | word |
| <i>decimal</i> | 0123456789 | digit | integer |
| <i>special</i> | ~`!@#\$%^&*()_-+={}[] \,:;''<,>.?/ | | |
| <i>keyboard</i> | <i>Roman + decimal + special</i> | keystroke | typescript |
| <i>genetic code</i> | ATCG | nucleotide base | DNA |
| <i>protein code</i> | ACDEFGHIJKLMNOPQRSTUVWXYZ | amino acid | protein |
| <i>ASCII</i> | see SECTION 6.1 | byte | String |
| <i>Unicode</i> | see SECTION 6.1 | char | String |

Commonly used alphabets and associated terminology

| <i>formal language</i> | <i>in the language</i> | <i>not in the language</i> |
|--------------------------------|--|--------------------------------------|
| <i>amino acid encodings</i> | AAA AAC AAG AAT ACA ACC ACG ACT TAC TAT TGC TGG TGT | TAA TAG TGA AAAAAAAAA ABCDE |
| <i>U.S. telephone number</i> | (609) 258-3000 (800) 555-1212 | (99) 12-12-12 2147483648 |
| <i>English words</i> | and middle computability | abc niether misunderestimate |
| <i>legal English sentences</i> | This is a sentence. I think I can. | xya a b.c.e?? Cogito ergo sum. |
| <i>legal Java identifiers</i> | a class \$xyz3_XYZ | 12 123a a((BC))* |
| <i>legal Java programs</i> | public class Hi { public static void main(String[] args) { } } | int main(void) { return 0; } |

Regular expressions. A *regular expression* (RE) is a string of symbols that specifies a formal language. We define what regular expressions are (and what they mean) recursively, using the union, concatenation, and closure operations on sets, along with parentheses for operator precedence. Specifically, every regular expression is either an *alphabet symbol*, specifying the singleton set containing that symbol, or composed from the following operations (where R and S are REs):

- *Union:* $R \mid S$, specifying the union of the sets R and S,
- *Concatenation:* RS , specifying the concatenation of the sets R and S,
- *Closure:* R^* , specifying the closure of the set R,
- *Parentheses:* (R) , specifying the same set as R.

Definition. A language is *regular* if and only if it can be specified by an RE.

Using Regular Expressions to Make Many Text Processing Problems Solvable Quickly and Easily

- You might wonder why we would ever need to use regular expressions so why learn about them?
- Here are some use cases:
 - Searching for text where we don't know the specific text we are looking for up front but we do know some rules or patterns in the text:
 - Search for an IP or MAC address in a log (e.g. web server access log)
 - Search for a 10 digit mobile number that may optionally be preceded with a country code
 - Searching where the length of the text to extract is not known beforehand:
 - Search for URLs that start with `http://` or `https://`

Using Regular Expressions to Make Many Text Processing Problems Solvable Quickly and Easily

- Generating tokens by splitting a given text on delimiters of a variable type and length
- Extracting text that lies between 2 or more search patterns
- Validating input from the user (account number, usernames, credit card number etc)
- Getting parts of a text with some properties such as repeated words
- Conversions to custom predefined formats: e.g. insert comma after every three digits in numbers or remove commas that occur in parentheses
- Doing a global search replace while skipping escaped characters

Pattern matching

Pattern matching problem. Is a given string an element of a given set of strings?

Example 1 (from computational biochemistry)

An amino acid is represented by one of the characters CAVLIMCRKHDENQSTYFWP.

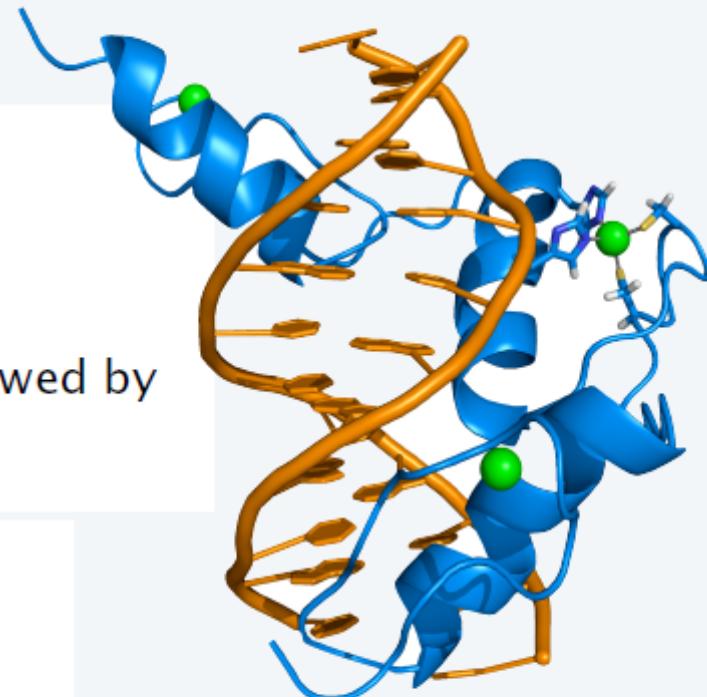
A protein is a string of amino acids.

A C₂H₂-type zinc finger domain signature is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by H.

Q. Is this protein in the C₂H₂-type zinc finger domain?

A. Yes.



Pattern matching

Example 2 (from commercial computing)

An e-mail address is

- A sequence of letters, followed by
- the character "@", followed by
- followed by a nonempty sequence of lowercase letters, followed by the character "."
- [any number of occurrences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Q. Which of the following are e-mail addresses?

Oops, need to fix description →

| | A. |
|------------------------|----|
| rs@cs.princeton.edu | ✓ |
| not an e-mail address | ✗ |
| wayne@cs.princeton.edu | ✓ |
| eve@airport | ✗ |
| rs123@princeton.edu | ✗ |

Challenge. Develop a precise description of the set of strings that are legal e-mail addresses.

you could search online

Pattern matching

Example 3 (from genomics)

A nucleic acid is represented by one of the letters a, c, t, or g.

A genome is a string of nucleic acids.

A **Fragile X Syndrome pattern** is a genome having an occurrence of gcg, followed by any number of cgg or agg triplets, followed by ctg.

Note. The number of triplets correlates with Fragile X Syndrome, a common cause of mental retardation.

Q. Does this genome contain a such a pattern?

gcggcgtgtgtcgagagagttaaagctggcgaggcggctggcgaggctg

A. Yes.

gcu start mark

ctt end mark

sequence of cgg and agg triplets

gcg cgg agg cgg ctg

Regular Expressions

Clever "wild card" expressions for matching and parsing strings.

Regular Expressions are an extended pattern matching language, they correspond to Finite State Automata

Regex Example: matching an IP address

- The regular expression `\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b` will match the following:
 - 110.65.147.246
 - 999.999.999.999
 - ۱۲۳.۹۲۳.۰۹۲.۷۷۳
- Note: IP addresses should have numbers ranging from 0-255

Regex Example: matching an IP address

- This one will store each of the numbers in the ip address in a separate group (using ()), it will only match the numbers if they are in the range 0 – 255.

```
\b(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.
(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\b
```

- Note: “\b” matches a word boundary, “[0-5]” matches characters 1,2,3,4 or 5, “?” means non greedy – so match one or more of the proceeding pattern

Regex Tester (also for learning regex)

<https://regex101.com/>

<https://regrexr.com/>

You can test regex matching easily at these links (there are others too)

The screenshot shows the regex101.com website. At the top, there's a navigation bar with links for regular expressions, social media, donations, sponsorships, contact, bug reports, a wiki, and what's new. Below the bar, the main interface has two main sections: "REGULAR EXPRESSION" and "TEST STRING". The expression field contains the pattern `/ insert your regular expression here / gm`, resulting in "no match". The test string field contains the placeholder "insert your test string here". To the right, there are sections for "EXPLANATION", "MATCH INFORMATION", and "QUICK REFERENCE". The "EXPLANATION" section says "An explanation of your regex will be automatically generated as you type." The "MATCH INFORMATION" section says "Detailed match information will be displayed automatically." The "QUICK REFERENCE" section includes a search bar for tokens and a list of tokens: "A sing... [abc]" and "A cha... [^abc]".

The screenshot shows the regrexr.com website. The top navigation bar includes links for GitHub and Sign In. A sidebar on the left contains a "Menu" with options for Pattern Settings, My Patterns, Cheatsheet, and RegEx Reference. The main area has tabs for "Expression", "Text", and "Tests". The expression input field contains the pattern `/([A-Z])\w+/g`. The results panel shows "No match (0.1ms)". Below the results, there's a summary of RegExr's features: "RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp)." It lists four bullet points: "Supports JavaScript & PHP/PCRE RegEx.", "Results update in real-time as you type.", "Roll over a match or expression for details.", and "Validate patterns with suites of Tests." At the bottom, it says "Save & share expressions with others".

Expression

```
/\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b/g
```

Text

Tests

Digit. Matches any digit character (0-9).

```
110.65.147.246
```

<https://regexr.com/>

Tools

Roll-over elements below to highlight in the Expression above. Click to open in Reference.

\b Word boundary. Matches a word boundary position between a word character and non-word character or position (start / end of string).

\d Digit. Matches any digit character (0-9).

{1,3} Quantifier. Match between 1 and 3 of the preceding token.

\. Escaped character. Matches a "." character (char code 46).

\d Digit. Matches any digit character (0-9).

{1,3} Quantifier. Match between 1 and 3 of the preceding token.

\. Escaped character. Matches a "." character (char code 46).

Regular expression - Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Regular_expression

Reader Google

More than 100 matches regular Done

Log in / create account

Article Discussion Read Edit View history Search

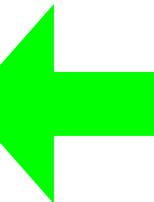
Regular expression

From Wikipedia, the free encyclopedia

In computing, a **regular expression**, also referred to as **regex** or **regexp**, provides a concise and flexible means for matching **strings** of text, such as particular characters, words, or patterns of characters. A regular expression is written in a **formal language** that can be interpreted by a regular expression processor, a program that either serves as a **parser generator** or examines text and identifies parts that match the provided specification.

The following examples illustrate a few specifications that could be expressed in a regular expression:

- The sequence of characters "car" appearing consecutively in any context, such as in "car", "cartoon", or "bicarbonate"
- The sequence of characters "car" occurring in that order with other characters between them, such as in "Icelander" or "chandler"



Really smart "Find" or "Search"

Understanding Regular Expressions

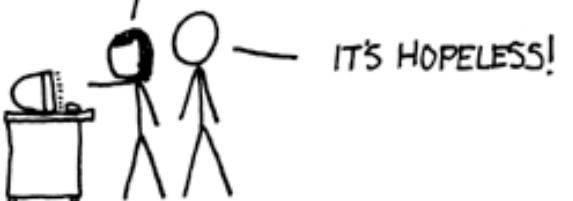
- Very powerful for text processing but cryptic
- Fun once you understand them
- Regular expressions are a language by themselves
- Programming with characters

WHENEVER I LEARN A
NEW SKILL I CONCOCT
ELABORATE FANTASY
SCENARIOS WHERE IT
LETS ME SAVE THE DAY.

OH NO! THE KILLER
MUST HAVE FOLLOWED
HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH
THROUGH 200 MB OF EMAILS LOOKING FOR
SOMETHING FORMATTED LIKE AN ADDRESS!



EVERYBODY STAND BACK.



I KNOW REGULAR
EXPRESSIONS.



<http://xkcd.com/208/>

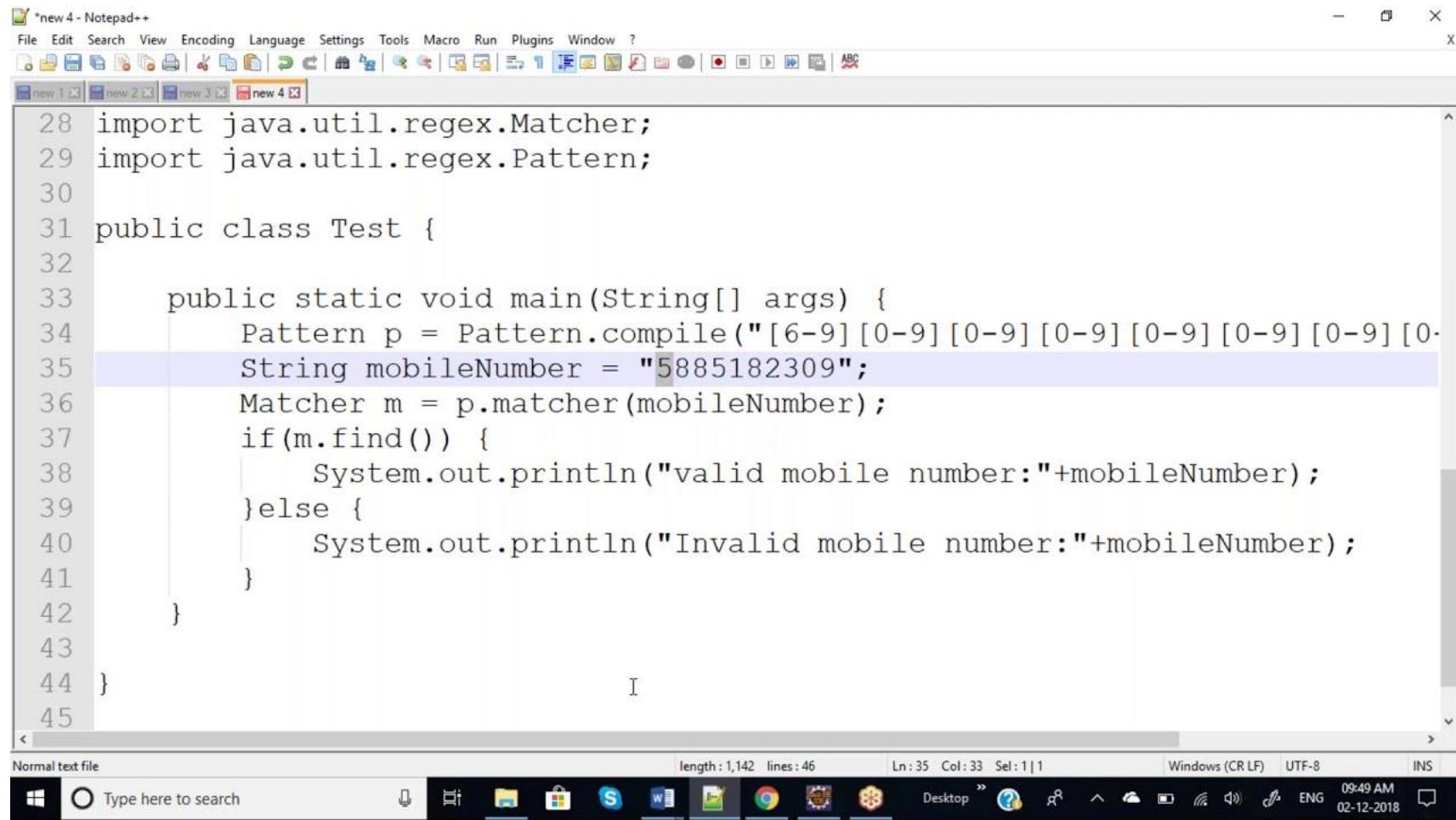
Regular Expression Quick Guide

- ^ Matches the **beginning** of a line
- \$ Matches the **end** of the line
- . Matches **any** character
- \s Matches **whitespace**
- \S Matches any **non-whitespace** character
- *
- Repeats a character zero or more times
- *?
- Repeats a character zero or more times (non-greedy)
- +
- Repeats a character one or more times
- +?
- Repeats a character one or more times (non-greedy)
- [aeiou] Matches a single character in the listed **set**
- [^XYZ] Matches a single character **not** in the listed **set**
- [a-z0-9] The set of characters can include a **range**
- (Indicates where string **extraction** is to start
-) Indicates where string **extraction** is to end

Using Regular Expressions in Your Programs

Before you can use regular expressions in your program, you must import the library `java.util.regex.Matcher` and `Pattern` libraries, they are used in a specific way in java, and regular expressions are used in many other programming languages (python, etc)

Java



The screenshot shows a Notepad++ window with Java code for validating a mobile number using regular expressions. The code defines a class Test with a main method that checks if a given mobile number matches the pattern [6-9][0-9][0-9][0-9][0-9][0-9][0-9]. The mobile number "5885182309" is hardcoded and passed to the matcher method. The output is printed to the console based on whether the find method returns true or false.

```
28 import java.util.regex.Matcher;
29 import java.util.regex.Pattern;
30
31 public class Test {
32
33     public static void main(String[] args) {
34         Pattern p = Pattern.compile("[6-9][0-9][0-9][0-9][0-9][0-9][0-9]");
35         String mobileNumber = "5885182309";
36         Matcher m = p.matcher(mobileNumber);
37         if(m.find()) {
38             System.out.println("valid mobile number:"+mobileNumber);
39         }else {
40             System.out.println("Invalid mobile number:"+mobileNumber);
41         }
42     }
43
44 }
45
```

Wild-Card Characters

- The dot character matches any character
- If you add the asterisk character, the character is "any number of times"

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-DSPAM-Confidence: 0.8475

X-Content-Type-Message-Body: text/plain

$\wedge X.*:$

Wild-Card Characters

- The **dot** character matches any character
- If you add the **asterisk** character, the character is "any number of times"

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-DSPAM-Confidence: 0.8475

X-Content-Type-Message-Body: text/plain

Match the start of the line

Many times

^X.*:



Match any character

Fine-Tuning Your Match

- Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-Plane is behind schedule: two weeks

Match the start of the line

Many times

`^X.*:`

Match any character

Fine-Tuning Your Match

- Depending on how "clean" your data is and the purpose of your application, you may want to narrow your match down a bit

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-Plane is behind schedule: two weeks

Match the start of the line

^X-\S+:

One or more times

Match any non-whitespace character

Matching and Extracting Data

We can extract the strings using () and group

[0-9]+



One or more digits

```
String x = "My 2 favorite numbers are 19 and 42";
String patS = "([0-9]+)";
Pattern p = Pattern.compile(patS );
Matcher m = p.matcher(x);
if (m.find())
    System.out.println(m.group(0)+" "+ m.group(1)+" "
                        m.group(2));
```

Warning about Greedy Matching

- The **repeat** characters (* and +) push **outward** in both directions (greedy) to match the largest possible string

String patS = "^\u0391F.+:";

String = "From: Using the : character"

Will match this

['From: Using the :]'

Why not 'From:'?

First character in the
match is an F

One or more
characters

^F.+:

Last character in the
match is a :

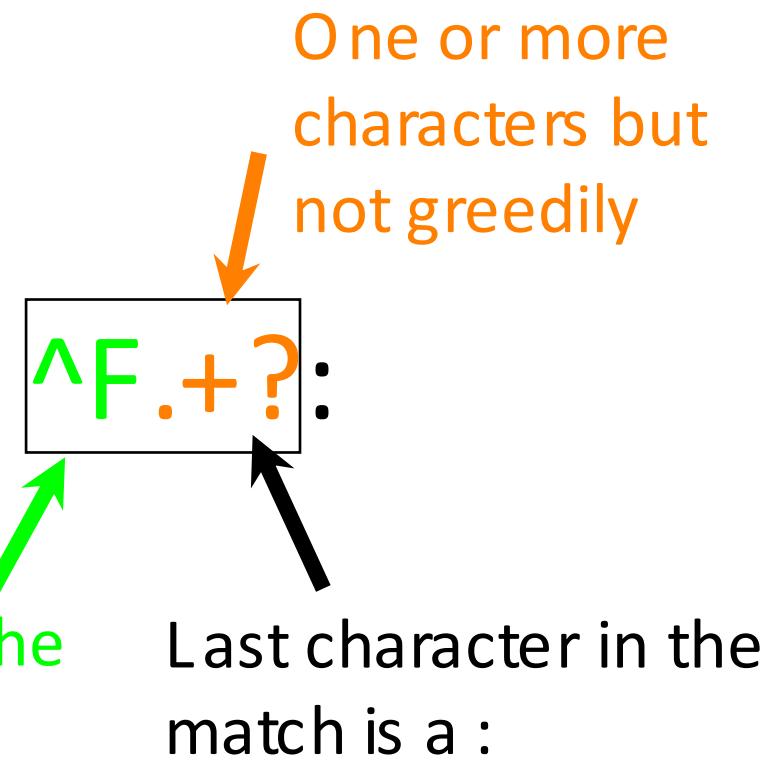
Non-Greedy Matching

- Not all regular expression repeat codes are greedy! If you add a ? character - the + and * chill out a bit...

Line: 'From: Using the : character'

Pattern '^F.+?:'

Matches: ['From:']



Regular Expression Quick Guide

- ^ Matches the **beginning** of a line
- \$ Matches the **end** of the line
- . Matches **any** character
- \s Matches **whitespace**
- \S Matches any **non-whitespace** character
- *
- Repeats a character zero or more times
- *?
- Repeats a character zero or more times (non-greedy)
- +
- Repeats a character one or more times
- +?
- Repeats a character one or more times (non-greedy)
- [aeiou] Matches a single character in the listed **set**
- [^XYZ] Matches a single character **not** in the listed **set**
- [a-z0-9] The set of characters can include a **range**
- (Indicates where string **extraction** is to start
-) Indicates where string **extraction** is to end

(Deterministic) Finite Automata

- May be used to define languages formally using:
 - A set of states
 - An alphabet
 - A transition function
 - A starting state
 - A final state (that is also called an accepting state)

Transition function

- Transition functions take 2 arguments: a state and an input symbol:

Formally:

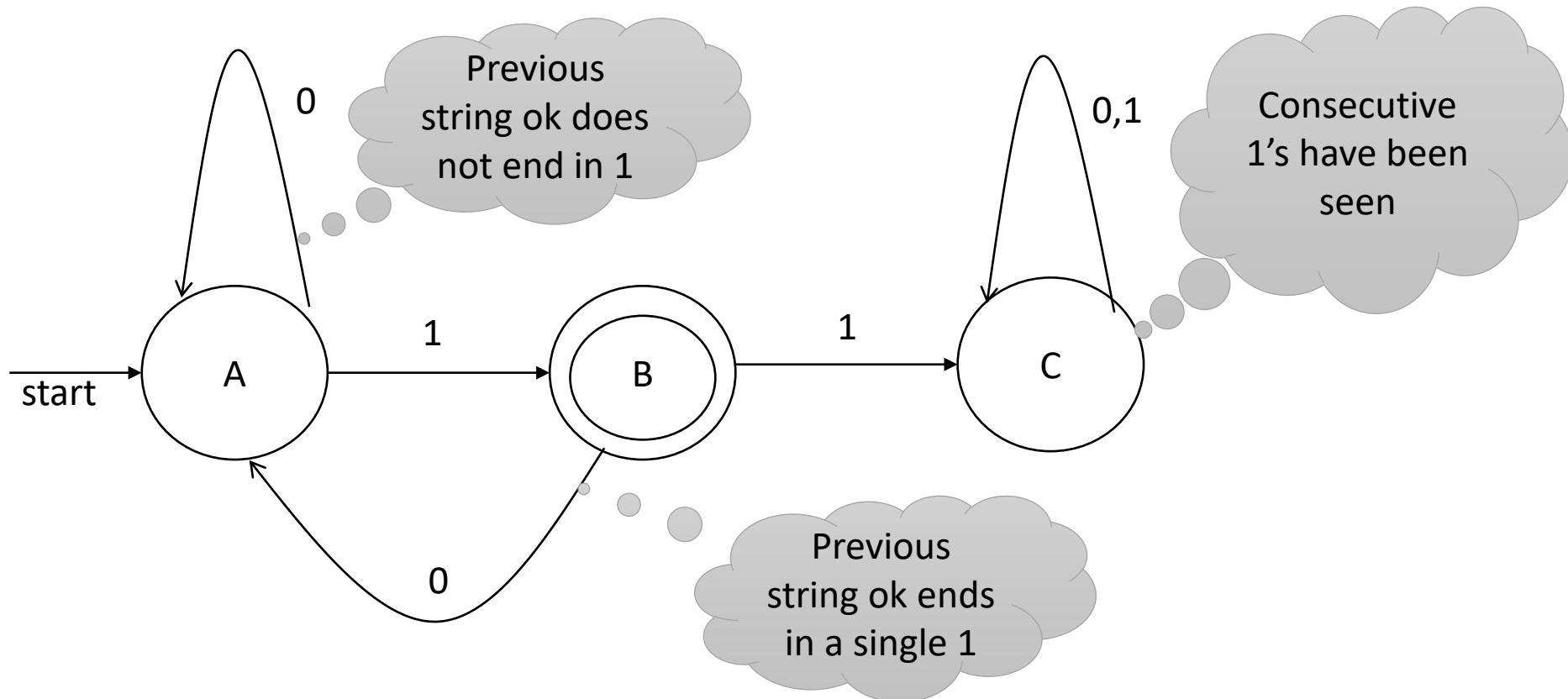
$$\delta(q, a) = q'$$

- δ will map the state q to another state q' given the input symbol a

FSM Graphs

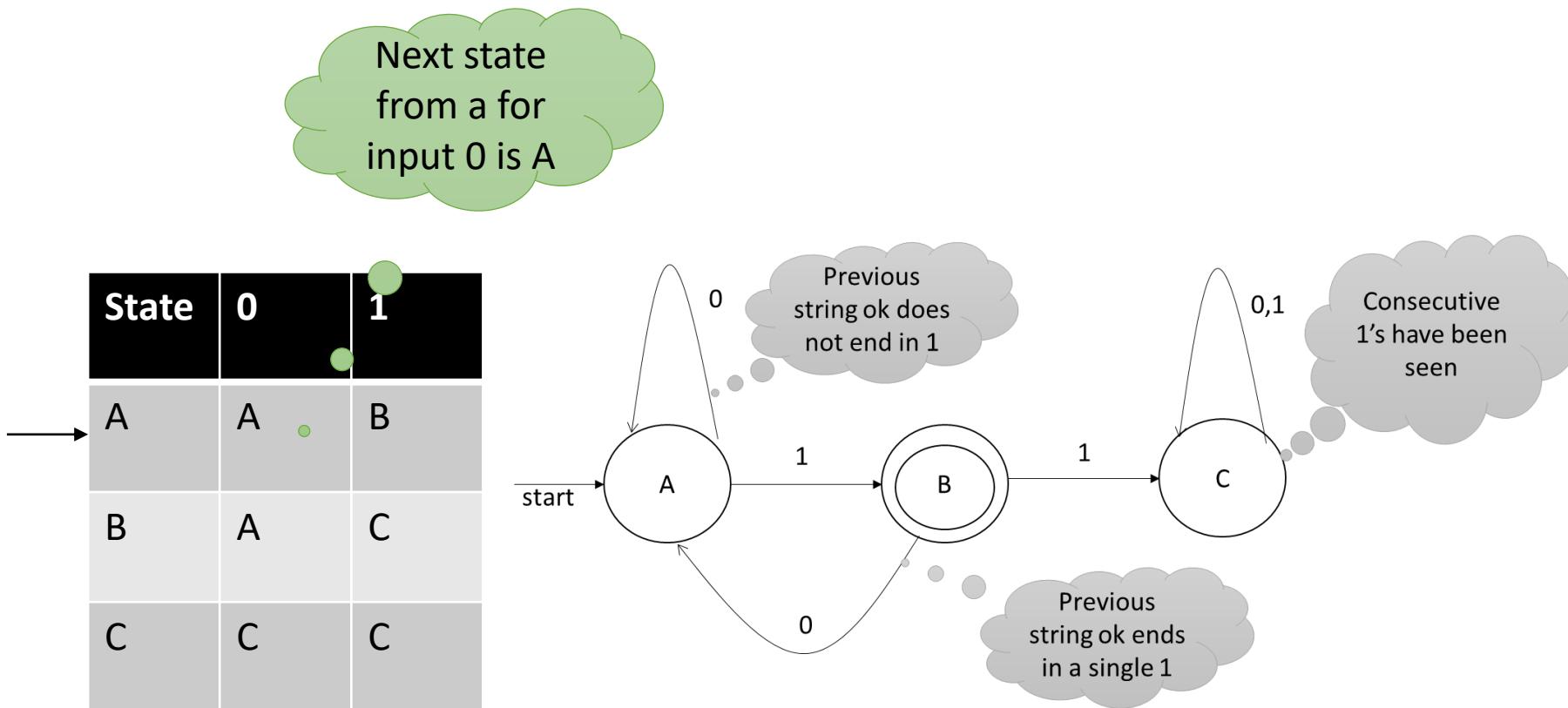
- Vertices represent states
- Edges are directed and represent a transition function
 - An edge from state q to q' is labelled by input symbols that have transitions from q to q'
- An arrow should be labelled start to indicate the starting state
- The final state can be marked by a double circle

Example for alphabet $\{0,1\}$



This FA accepts all strings without two consecutive 1's

Transition Tables



$$\begin{aligned}\delta(A, 0) &= A \\ \delta(A, 1) &= B \\ \delta(B, 0) &= A \\ \delta(B, 1) &= C \\ \delta(C, 0) &= C \\ \delta(C, 1) &= C\end{aligned}$$

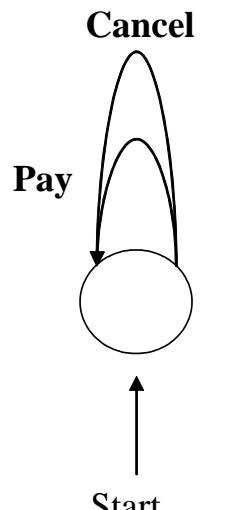
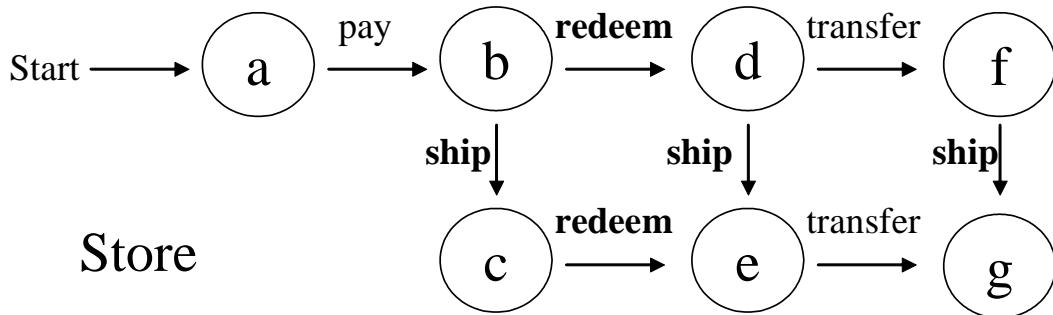
2 Types of Finite Automata

- Deterministic (DFA) – There is a fixed number of states and we can only be in one state at a time
- Nondeterministic (NFA) –There is a fixed number of states but we can be in multiple states at one time

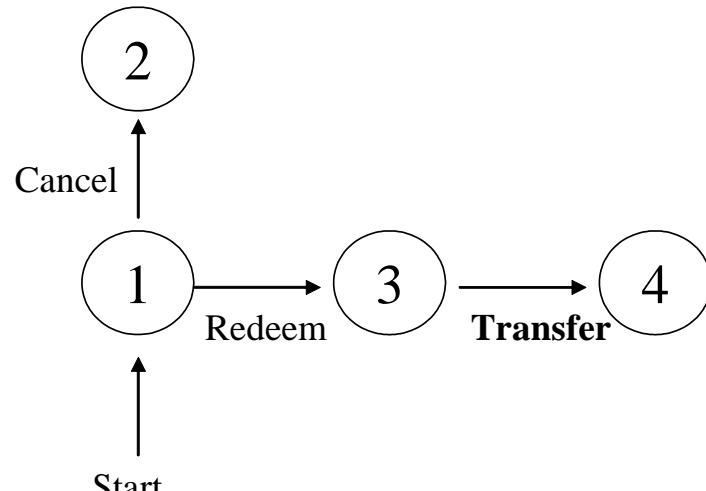
Many Problems Can Be Modelled With FA

- Customer shopping at a store with an electronic transaction with the bank
 - The customer may *pay* the e-money or *cancel* the e-money at any time.
 - The store may *ship* goods and *redeem* the electronic money with the bank.
 - The bank may *transfer* any redeemed money to a different party, say the store.
- Can model this problem with three automata

Bank Automata



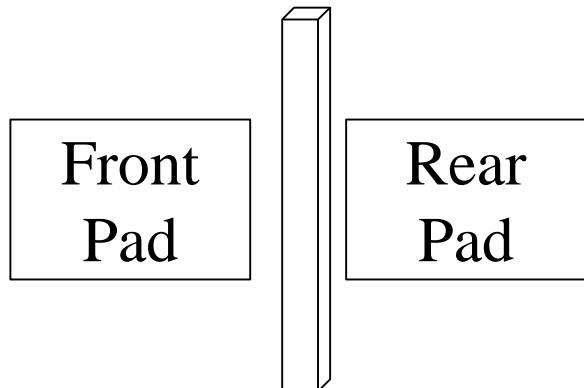
Customer



Bank

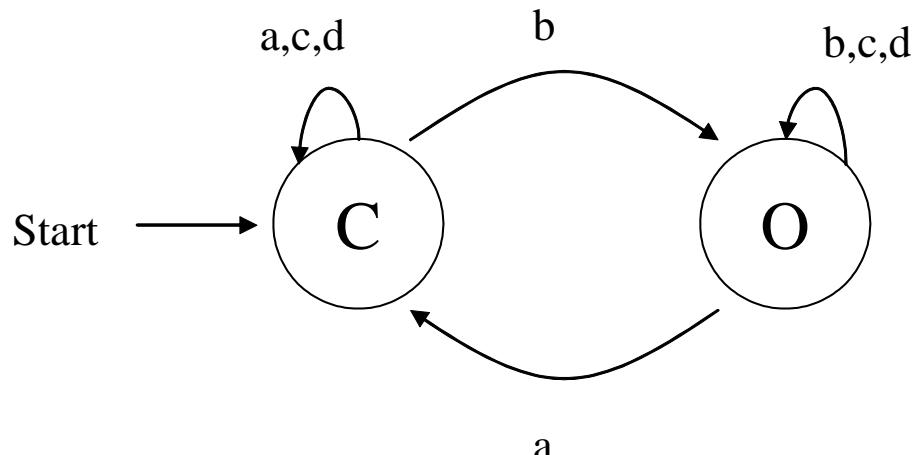
Simple Example – 1 way door

- As an example, consider a one-way automatic door. This door has two pads that can sense when someone is standing on them, a front and rear pad. We want people to walk through the front and toward the rear, but not allow someone to walk the other direction:



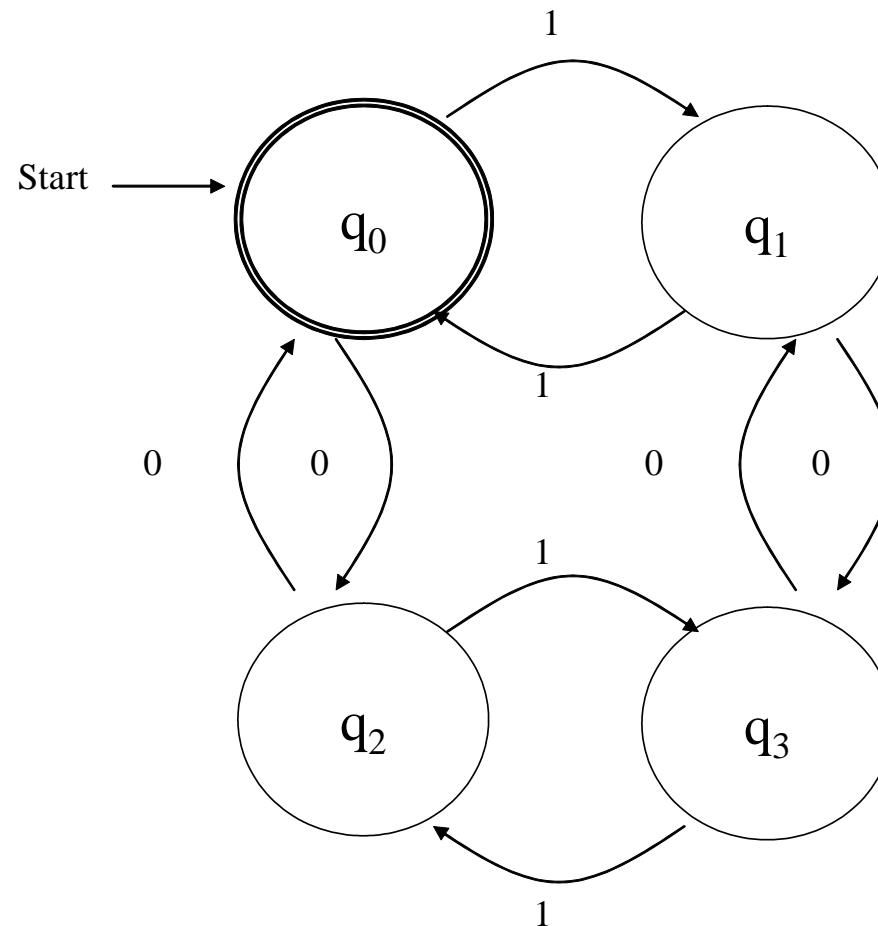
One Way Door

- Let's assign the following codes to our different input cases:
 - a - Nobody on either pad
 - b - Person on front pad
 - c - Person on rear pad
 - d - Person on front and rear pad
- We can design the following automaton so that the door doesn't open if someone is still on the rear pad and hit them:



DFA Example

Here is a DFA for the language that is the set of all strings of 0's and 1's whose numbers of 0's and 1's are both even



NFA

- Non Deterministic Finite Automata allow several states to be present at once
- Any NFA can be shown to have an equivalent to DFA
- We will not further discuss NFA in this course

Matching with Finite Automata

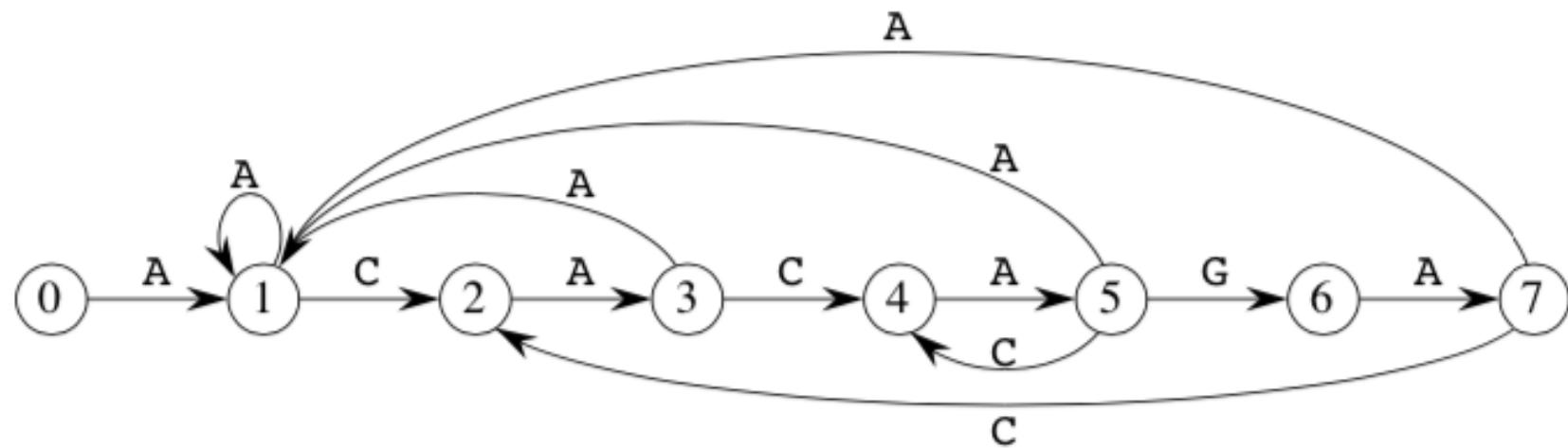
- A ***finite automaton (FA)*** is a set of states and a way to go from state to state based on a sequence of input characters. An FA starts in a given state and “consumes” characters one at a time. Based on the state it’s in and the character just consumed, it moves to a new state.

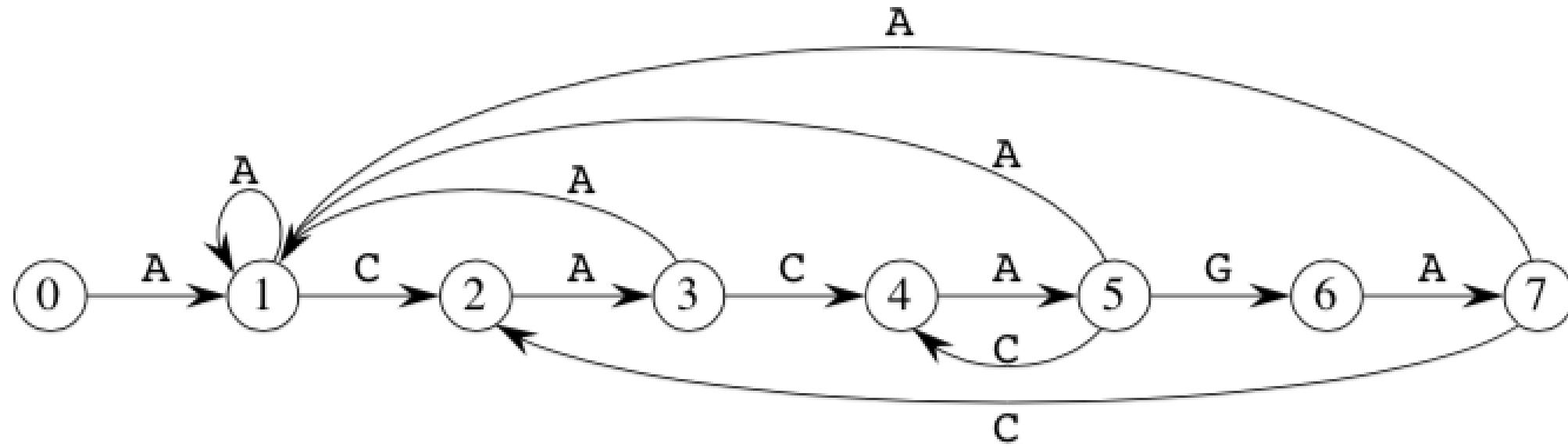
Formally: A finite automaton M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$ where

- Q is a finite set of ***states***,
- $q_0 \in Q$ is the ***start state***,
- $A \subseteq Q$ is a distinguished set of ***accepting states***,
- Σ is a finite ***input alphabet***,
- δ is a function $Q \times \Sigma \rightarrow Q$, called the ***transition function*** of M .

FA Example

- Consider DNA sequencing where an alphabet $\Sigma = \{A, C, G, T\}$
- Pattern P = ACACAGA
- The Finite Automata is:





- The horizontal main line has edges labelled with P
- Whenever P occurs in the text the FA moves right to the next state
- When it reaches the far right hand state we have a match
- Note: the transition function δ (delta) is defined for all states $q \in Q$ and all characters $a \in \Sigma$ (in the alphabet) and missing arrows are assumed to be transitions to state 0

FA-MATCHER(T, δ, n, m)

$q = 0$

for $i = 1$ **to** n

$q = \delta(q, T[i])$

if $q == m$

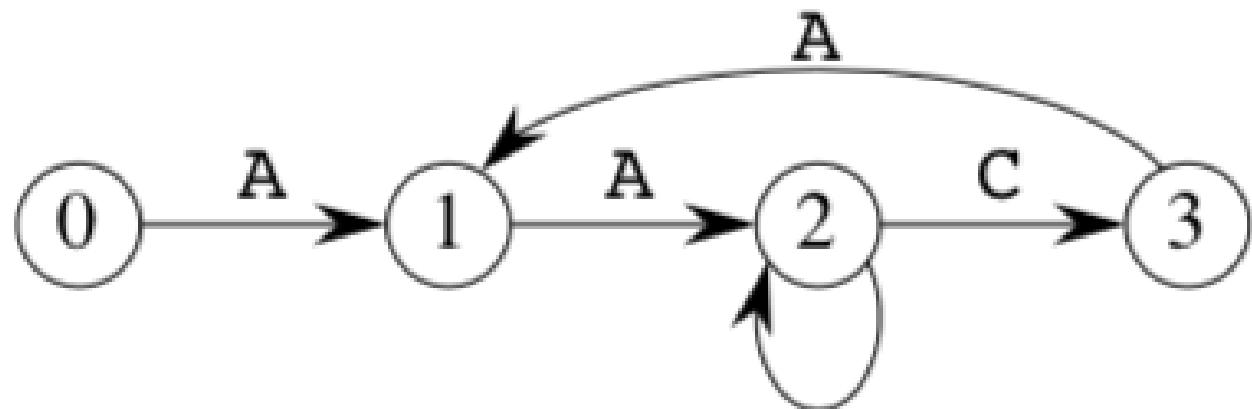
 print “Pattern occurs with shift” $i - m$

Example 2

Sequence of states, input characters, and output shifts:

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|--------------|---|---|---|---|---|---|---|---|---|----|----|----|----|
| character | G | T | A | A | C | A | G | T | A | A | A | C | G |
| state | 0 | 0 | 0 | 1 | 2 | 3 | 1 | 0 | 0 | 1 | 2 | 2 | 3 |
| output shift | | | | | | | 2 | | | | | | 9 |

Example: $P = AAC$, $T = GTAACACAGTAAACG$. FA is

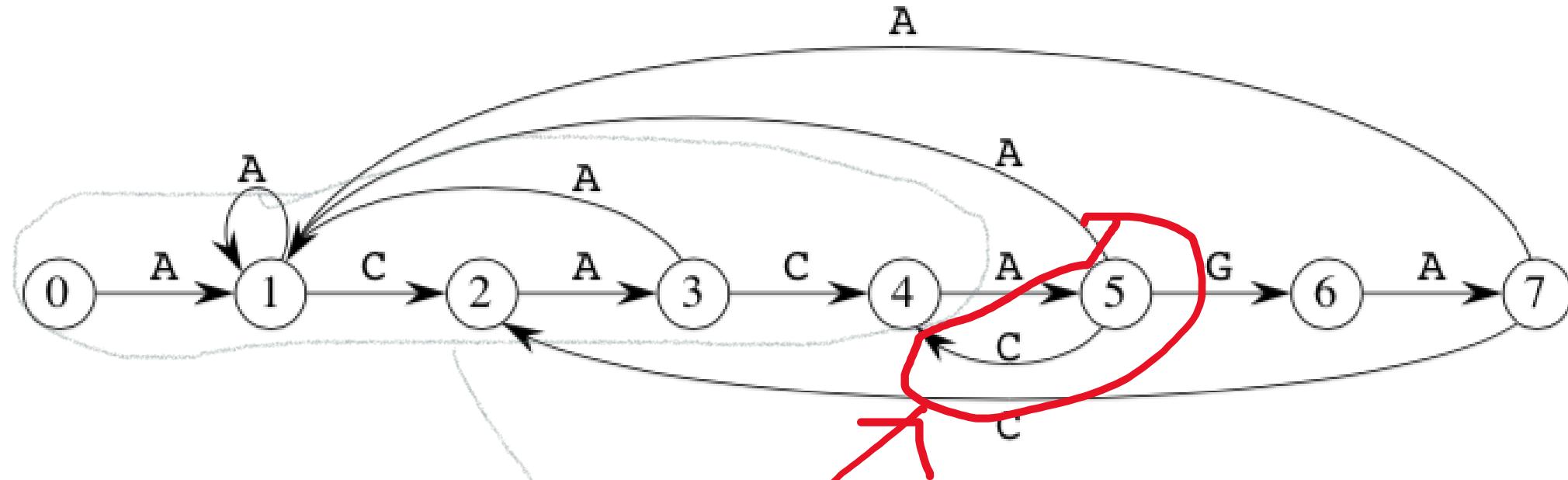


$\Theta(n)$

The pre-process to build the Finite Automaton

- We know it has states $0 .. M$ (the length of the pattern)
- We know the starting state is 0
- We “just” need to determine the transition function δ
- **Basic idea:** when the FA is in state k , the most recent characters read from the text are the first k characters in the pattern

The FA of the pattern ACACAGA:

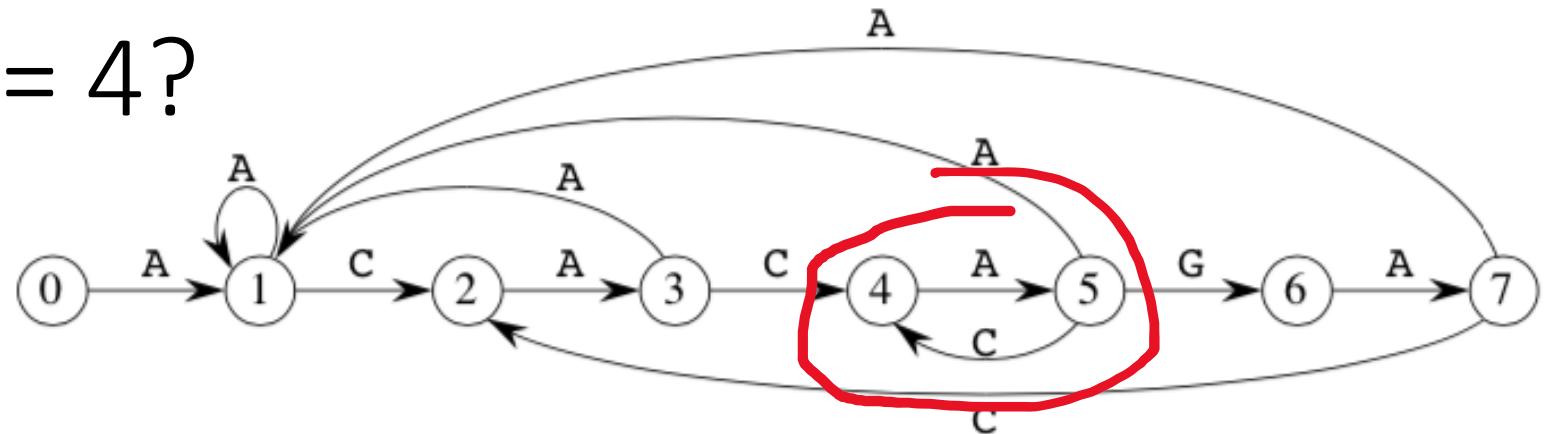


Why is $\delta(5, c) = 4$?

If the FA gets to state 5 then the 5 most recent characters read are ACACA (see the “spine”)

If the next character read is C then it doesn't match so the FA cannot go to state 6. But it does not need to go back to state 0 either because now the 4 most recently read characters are ACAC: first 4 characters of the pattern so the FA moves to state 4

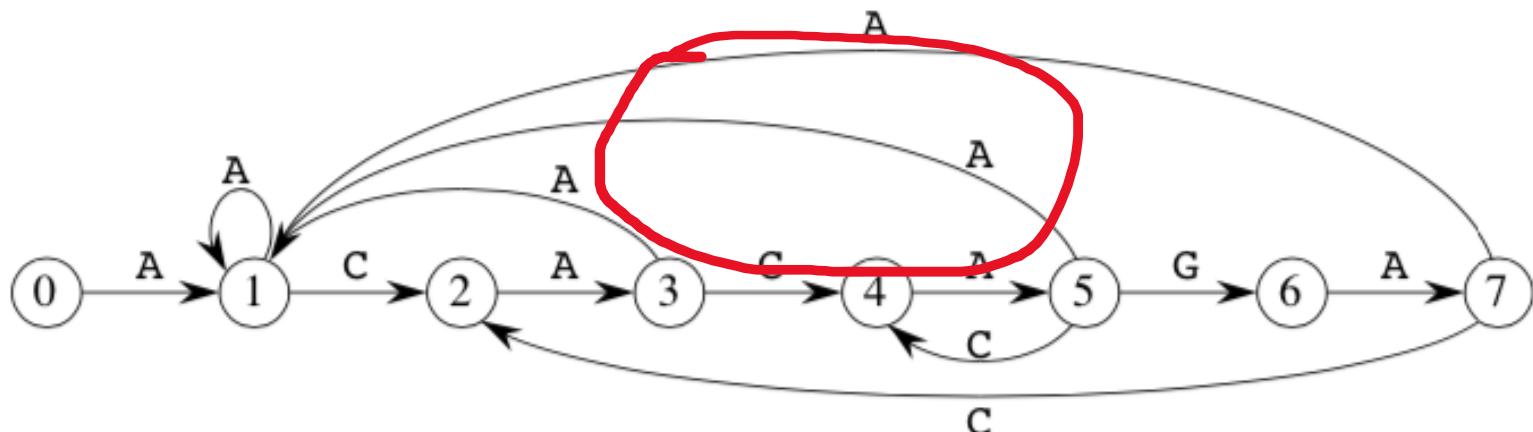
Why is $\delta(5,c) = 4$?



- The **prefix** P_i of a string P is a substring consisting of the first i characters of P
- The **suffix** is a substring consisting of characters from the end
- For a string X and a character a denote concatenating a onto X by Xa
- In state k we most recently read P_k in the text
- We look at the next character a of the text so now we've read $P_k a$
- How long a prefix of P have we just read:

Find the longest prefix of P that is also a suffix of $P_k a$. Then $\delta(k,a)$ should be the length of the longest prefix

Why is $\delta(5, c) = 4$?



- Take the prefix $P_5 = \text{ACACA}$.
- Concatenate **C**, giving **ACACAC**.
- Want the longest prefix of the pattern **ACACAGA** that is also a suffix of **ACACAC**.
- Because the length of **ACACAC** is 6 and the suffix can't be longer than that, start by looking at P_6 . Work our way down to shorter and shorter prefixes until we find a prefix that is also a suffix of **ACACAC**.
- P_6 is **ACACAG**. Not a suffix of **ACACAC**.
- P_5 is **ACACA**. Not a suffix of **ACACAC**.
- P_4 is **ACAC**. This is a suffix of **ACACAC**, so stop and set $\delta(5, \mathbf{C}) = 4$.

COMPUTE- $\delta(P, \Sigma, m)$

for $q = 0$ **to** m

for each character $a \in \Sigma$

$k = \min(m + 1, q + 2)$

repeat

$k = k - 1$

until P_k is a suffix of $P_q a$

 set $\delta(q, a) = k$

return δ

We're guaranteed to stop, because P_0 is the empty string, and the empty string is a suffix of any string. So if we don't find a suffix of $P_k a$ from among $P_k, P_{k-1}, P_{k-2}, \dots, P_1$, then set $\delta(k, a) = 0$.

Because k is decremented in the **repeat** loop before the first suffix test, its maximum value is $\min(m, q + 1)$. If we set $\delta(q, a) = q + 1$, then P_{q+1} is a suffix of $P_q a$, and so we have found the next character in the pattern. We cap k at m , since we can't go to a state numbered higher than m .

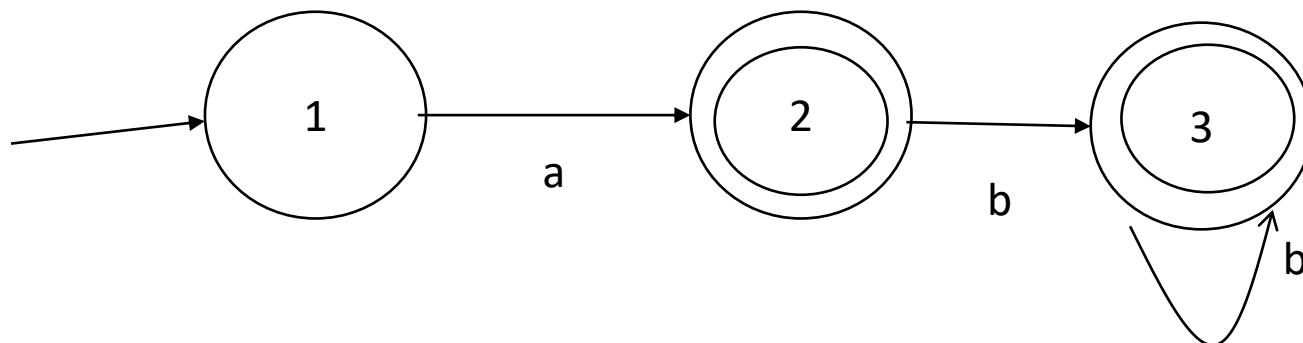
Preprocessing time:

- The outermost **for** loop iterates $m + 1$ times.
- The middle loop iterates $|\Sigma|$ times.
- The **repeat** loop iterates at most $m + 1$ times.
- Each suffix check in the **until** part examines at most m characters of the pattern (since $k \leq m$). Therefore, each suffix check takes $O(m)$ time.
- Total preprocessing time: $O(m^3 |\Sigma|)$.

Therefore, with a finite automaton, we can perform string matching with preprocessing time $O(m^3 |\Sigma|)$ and matching time $\Theta(n)$. It's possible to get the preprocessing time down to $O(m |\Sigma|)$.

DFA and Regular Expressions

- Regular Expressions can be represented with DFA, for example:
- ab^* means any string starting with a and then and then 0 or more b



- **Note:** Here the meaning of the wildcard * could be different from regular expressions in some programming languages. Sometimes the regex used in programming languages uses * to represent any number of any character. Here it is formally used to indicate any number of just the preceding character.

Regular expressions

A regular expression (RE) is a notation for specifying a set of strings (a formal language).

An RE is either

- The empty set
- The empty string
- A single character or wildcard symbol
- An RE enclosed in parentheses
- The *concatenation* of two or more REs
- The *union* of two or more REs
- The *closure* of an RE
(any number of occurrences)

| <i>operation</i> | <i>example RE</i> | <i>matches</i> (IN the set) | <i>does not match</i> (NOT in the set) |
|----------------------|---------------------|----------------------------------|---|
| <i>concatenation</i> | aabaab | aabaab | <i>every other string</i> |
| <i>wildcard</i> | .u.u.u. | cumulus jugulum | succubus tumultuous |
| <i>union</i> | aa baab | aa baab | <i>every other string</i> |
| <i>closure</i> | ab*a | aa abbba | ab ababa |
| <i>parentheses</i> | a(a b)aab (ab)*a | aaaab abaab a ababababa | <i>every other string</i> |

More examples of regular expressions

The notation is surprisingly expressive.

| <i>regular expression</i> | <i>matches</i> | <i>does not match</i> |
|---|---|-------------------------------------|
| $\cdot^* \text{spb} \cdot^*$ <i>contains the trigraph spb</i> | raspberry crispbread | subspace subspecies |
| $a^* \mid (a^*ba^*ba^*ba^*)^*$ <i>multiple of three b's</i> | bbb aaa bbbaababbaa | b bb baabbbaa |
| $\cdot^* 0 \dots$ <i>fifth to last digit is 0</i> | 1000234 98701234 | 111111111 403982772 |
| $\cdot^* \text{gcg}(\text{cg}\text{g} \mid \text{agg})^* \text{ctg} \cdot^*$ <i>fragile X syndrome pattern</i> | ...gcgctg... ...gcgcggctg... ...gcgcggaggctg... | gcgcg cggcggcggctg gcgcaggctg |

Generalized regular expressions

Additional operations further extend the utility of REs.

| operation | example RE | matches | does not match |
|-----------------|-------------------------|---|---------------------------|
| one or more | $a(bc)^+de$ | abcde abcbcde | ade bcde |
| character class | $[A-Za-z][a-z]^*$ | lowercase Capitalized | camelCase 4illegal |
| exactly j | $[0-9]\{5\}-[0-9]\{4\}$ | 08540-1321 19072-5541 | 1111111111 166-54-1111 |
| between j and k | $a.\{2,4\}b$ | abcb abcacb | ab aaaaaab |
| negation | $[\^aeiou]\{6\}$ | rhythm | decade |
| whitespace | \s | any whitespace char (space, tab, newline...) | every other character |

Note. These operations are all *shorthand*.
They are very useful but not essential.

RE: $(a|b|c|d|e)(a|b|c|d|e)^*$
shorthand: $(a-e)^+$

Example of describing a pattern with a generalized RE

A C₂H₂-type zinc finger domain signature is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by



Q. Give a generalized RE for all such signatures.

A. C . {2 , 4} C . . . [LIVMFYWCX] . {8} H . {3 , 5} H

"Wildcard" matches any of the letters
CAVLIIMCRKHDENQSTYFWP



Constructs of the standard regular expression and meta characters

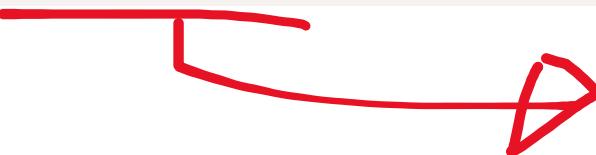
Let's get familiar with core constructs of regular expressions and some reserve meta characters that have a special meaning in regular expressions.

| Symbol | Meaning | Example |
|-------------------|--|---|
| • (dot or period) | Matches any character other than newline. | Matches #, @, A, f, 5, or . |
| * (asterisk) | * matches zero or more occurrences of the preceding character or group. | m* matches 0 or more occurrences of the letter m. |
| + | + matches one or more occurrences of the preceding element. | m+ matches one or more occurrences of the letter m. |
| ? (question mark) | ? means optional match. It is used to match zero or one occurrence of the preceding element. It is also used for lazy matching (which will be covered in the coming chapters). | nm? means match n or nm, as m is an optional match here. |

Lazy and Greedy Matching

- Quantifiers are simple but can be confusing
- When looking for a match the reg ex evaluator does this:
 - For every position in the string
 - Try to match the pattern at that position.
 - If there's no match, go to the next position.
- Greedy matching quantifiers, such as “**.**” attempt to match as many instances of a pattern as possible in a string
- Lazy matching using “**?**” (e.g. “**.**?” or “***?**”) changes this behavior to match the minimal number of times

```
let regexp = /.+$/g;  
  
let str = 'a "witch" and her "broom" is one';  
  
alert( str.match(regexp) ); // "witch" and her "broom"
```



returns the whole string:
"witch" and her "broom"

```
let regexp = /.+?$/g;  
  
let str = 'a "witch" and her "broom" is one';  
  
alert( str.match(regexp) ); // witch, broom
```



now it returns the two matching strings:
"witch", "broom"

| Symbol | Meaning | Example |
|-------------|---|---|
| (pipe) | means alternation. It is used to match one of the elements separated by | <code>m n p</code> means match either the letter <code>m</code> or the letter <code>n</code> or the letter <code>p</code> |
| ^ (cap) | ^ is called anchor, that matches start of the line | <code>^m</code> matches <code>m</code> only when it is the first character of the string that we are testing against the regular expression. Also, note that you do not use <code>^</code> in the middle of a regular expression. |
| \$ (dollar) | \$ is called anchor that matches line end. | <code>m\$</code> matches <code>m</code> only at line end. |

| Symbol | Meaning | Example |
|--|--|--|
| \b (backslash followed by the letter b) | Alphabets, numbers, and underscore are considered word characters. \b asserts word boundary, which is the position just before and after a word. | \bjava\b matches the word, java . So, it will not match javascript since the word, javascript, will fail to assert \b after java in the regex. |
| \B (backslash followed by uppercase B) | \B asserts true where \b doesn't, that is, between two word characters. | For the input text, abc, \B will be asserted at two places: <ol style="list-style-type: none">1. Between a and b.2. Between b and c. |
| (...) a sub-pattern inside round parentheses | This is for grouping a part of text that can be used to capture a certain substring or for setting precedence. | \m(ab)*t matches m, followed by zero or more occurrences of the substring, ab, followed by t. |
| {min,max} | A quantifier range to match the preceding element between the minimum and the maximum number. | \mp{2,4} matches m followed 2 to 4 occurrences of the letter p. |

| Symbol | Meaning | Example |
|--|---|---|
| [...] | This is called a character class. | [A-Z] matches any uppercase English alphabet. |
| \d (backslash followed by the letter d) | This will match any digit. | \d matches any digit in the 0-9 range. |
| \D (backslash followed by uppercase D) | This matches any character that is not a digit. | \D matches a, \$, or _. |
| \s (backslash followed by the letter s) | Matches any whitespace, including tab, space, or newline. | \s matches [\t\n]. |
| \S (backslash followed by uppercase S) | Matches any non-whitespace. | \S matches the opposite of \s |

| Symbol | Meaning | Example |
|--|---|--|
| \w (backslash followed by the letter w) | Matches any word character that means all alphanumeric characters or underscore. | \w will match [a-zA-Z0-9_], so it will match any of these strings: "abc", "a123", or "pq_12_ABC" |
| \W (backslash followed by the letter W) | Matches any non-word character, including whitespaces. In regex, any character that is not matched by \w can be matched using \W. | It will match any of these strings: "+/=", "\$", or " !~" |

Some basic regular expression examples

- | ab*c
 - This will match a, followed by zero or more b, followed by c.
- | ab+c
 - This will match a followed by one or more b, followed by c.
- | ab?c
 - This will match a followed by zero or one b, followed by c.
 - Thus, it will match both abc or ac.
- | ^abc\$
 - This will match abc in a line, and the line must not have anything other than the string abc due to the use of the start and end anchors on either side of the regex.
- | a(bc)*z
 - This will match a, followed by zero or more occurrences of the string bc, followed by z. Thus, it will match the following strings: az, abc_z, abc_bc_z, abc_bc_bc_z, and so on.

| a(bc)*z

This will match a, followed by zero or more occurrences of the string bc, followed by z. Thus, it will match the following strings: az, abc_z, abc_bc_z, abc_bc_bc_z, and so on.

— | ab{1,3}c

This will match a, followed by one to three occurrences of b, followed by c. Thus, it will match following strings: abc, abbc, and abbbc.

— | red|blue

This will match either the string red or the string blue.

— | \b(cat|dog)\b

This will match either the string cat or the string dog, ensuring both cat and dog must be complete words; thus, it will **fail** the match if the input is cats or dogs.

— | [0-9]

This is a character class with a character range. The preceding example will match a digit between 0 and 9.

| [a-zA-Z0-9]

This is a character class with a character range. The preceding example will match any alpha-numeric character.

| ^\d+\$

This regex will match an input containing only one or more digits.

| ^\d{4,8}\$

This regex will allow an input containing four to eight digits only. For example, 1234, 12345, 123456, and 12345678 are valid inputs.

| ^\d\w\d\$

This regex not only allows only one digit at the start and end but also enforces that between these two digits there must be one non-digit character. For example, 1-5, 3:8, 8x2, and so on are valid inputs.

| ^\d+\.\d+\$

This regex matches a floating point number. For example, 1.23, 1548.567, and 7876554.344 are valid inputs.

| .+

This matches any character one or more times. For example, qwqewe, 12233, or f5^h_=!bg are all valid inputs:

| ^\w+\s+\w+\$

This matches a word, followed by one or more whitespaces, followed by another word in an input. For example, hello word, John smith, and United Kingdom will be matched using this regex.



Database of protein domains, families and functional sites



SARS-CoV-2 relevant PROSITE motifs

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [[More...](#) / [References](#) / [Commercial users](#)].

PROSITE is complemented by [ProRule](#), a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [[More...](#)].

Release 2020_02 of 22-Apr-2020 contains 1858 documentation entries, 1311 patterns, 1277 profiles and 1301 ProRule.

Search

e.g. [PDOC00022](#), [PS50089](#), [SH3](#), zinc finger

Type a regular expression here

Browse

- by documentation entry
- by ProRule description
- by taxonomic scope
- by number of positive hits

Quick Scan mode of ScanProsite

Other tools

Advanced RegEx

The effect of eager matching on regular expression alternation

This regular expression engine behavior may return unexpected matches in alternation if alternations are not ordered carefully in the regex pattern.

Take an example of this regex pattern, which matches the strings `white` or `whitewash`:

```
| white|whitewash
```

While applying this regex against an input of `whitewash`, the regex engine finds that the first alternative `white` matches the `white` substring of the input string `whitewash`, hence, the regex engine stops proceeding further and returns the match as `white`.

Note that our regex pattern has a better second alternative as `whitewash`, but due to the regex engine's eagerness to complete and return the match, the first alternative is returned as a match and the second alternative is ignored.

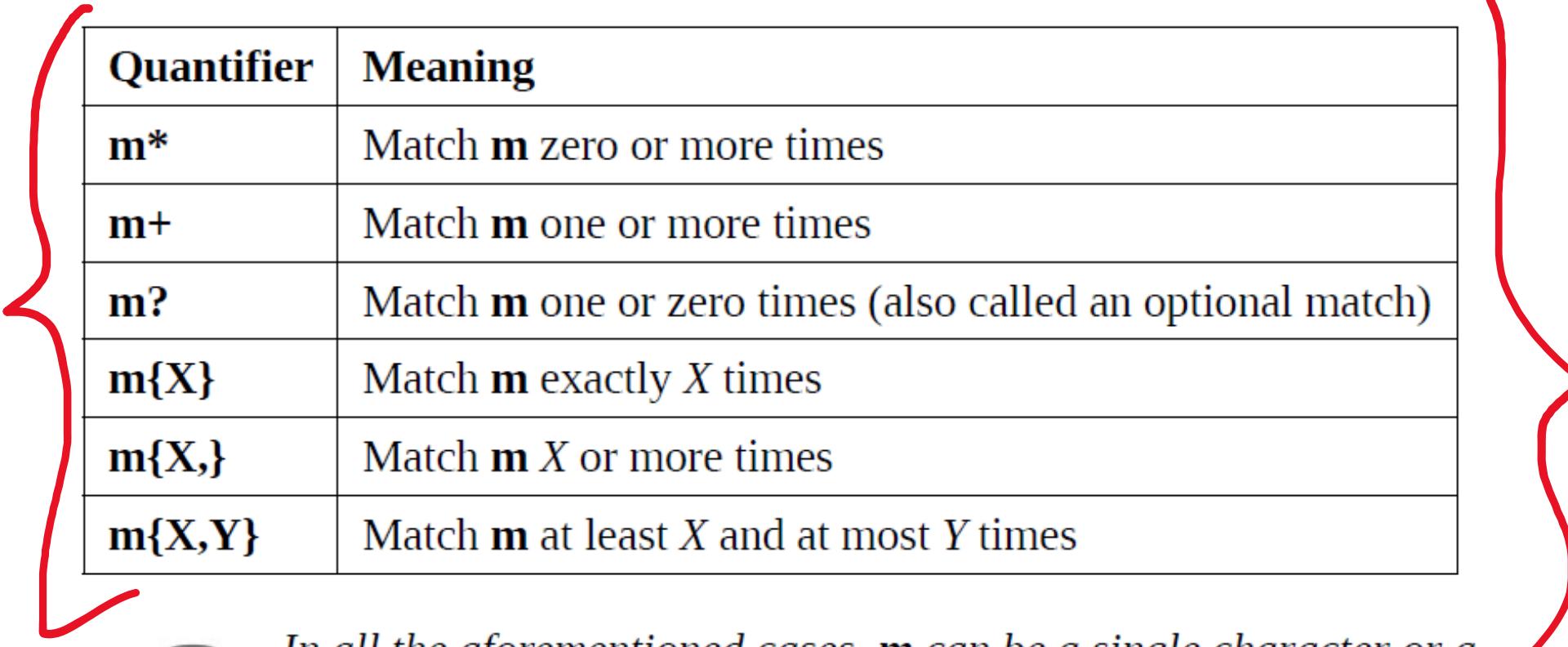
However, consider swapping the positions of the third and fourth alternatives in our regex pattern to make it as follows:

```
| whitewash|white
```

If we apply this against the same input, `whitewash`, then the regex engine correctly returns the match as `whitewash`.

Basic quantifiers

The following table lists all the quantifiers available in Java regular expressions:



| Quantifier | Meaning |
|------------|--|
| m^* | Match m zero or more times |
| m^+ | Match m one or more times |
| $m^?$ | Match m one or zero times (also called an optional match) |
| $m\{X\}$ | Match m exactly X times |
| $m\{X,\}$ | Match m X or more times |
| $m\{X,Y\}$ | Match m at least X and at most Y times |



*In all the aforementioned cases, **m** can be a single character or a group of characters. We will discuss grouping in more detail later.*

Examples using quantifiers

Let's look at few examples to understand these basic quantifiers better.

Which regex pattern should be used to match a two-digit year or a four-digit year?

| \d{2} | \d{4}



Which regex pattern should be used to match a signed decimal number? The pattern should also match a signed integer number:

| ^[+-]?\d*\.\d+\$



Here is the breakup of the preceding regex pattern:

- The `^` and `$` symbols are the start/end anchors
- The `[+-]?` pattern makes either the `+` sign or the `-` sign (optional because of `?`) at the start
- The `\d*` pattern matches zero or more digits
- The `\.\?` pattern matches an optional dot `(.)` literally
- The `\d+` pattern matches one or more digits

The preceding regex will match all of these inputs:

- .45
- 123789
- 5
- 123.45
- +67.66
- -987.34

| `^[-+]?[0-9]*[.][0-9]+$`

What would be the regex to match a number that is at least 10 but not more than 9999?

| `^\d{2,4}$`

Since we have a minimum of two digits, 10 is the smallest match, whereas the maximum number of digits allowed is four, and hence, 9999 is the highest match.

What is the regex for an input that has seven digits and that can have + or - at the start?

| `^[-+]?[0-9]{7}$`

The `[+-]?` pattern makes it an optional match at the start before we match the seven digits using `\d{7}`.

The preceding regex can also be written as `^[-+]?[0-9]{7}$`, as `\d` is a shorthand property to match `[0-9]`

If confusing try it here: <https://regex101.com>

SAVE & SHARE

[Save Regex](#) `⌘+S`

FLAVOR

</> PCRE (PHP) ✓
</> ECMAScript (JavaScript)
</> Python
</> Golang

TOOLS

[Code Generator](#)
[Regex Debugger](#)

SPONSOR

Try Azure for free

REGULAR EXPRESSION

`/ \d{4}|\d{2} / gm` 2 matches, 15 steps (~5ms)

TEST STRING

1920
20
|

[SWITCH TO UNIT TESTS](#)

EXPLANATION

▼ / `\d{4}|\d{2}` / gm
 ▼ 1st Alternative `\d{4}`
 ▼ `\d{4}` matches a digit (equal to [0-9])
 {4} Quantifier — Matches exactly 4 times
 ▼ 2nd Alternative `\d{2}`
 ▼ `\d{2}` matches a digit (equal to [0-9])
 {2} Quantifier — Matches exactly 2 times
— Global pattern flags

MATCH INFORMATION

Match 1
Full match 0-4 1920

Match 2
Full match 5-7 20

QUICK REFERENCE

Search reference A single c... [abc]
All Tokens A charac... [^abc]
Common Tokens A character... [a-z]
A character¹⁸¹ [^a-z]

Possessive quantifiers

Possessive quantifiers are quantifiers that are greedy when matching text like greedy quantifiers do. Both greedy and possessive quantifiers try to match as many characters as possible. The important difference, however, is that the possessive quantifiers do not backtrack (go back) unlike greedy quantifiers; therefore, it is possible that the regex match fails if the possessive quantifiers go too far.

This table shows all the three types of quantifiers, side by side:

| Greedy Quantifier | Lazy Quantifier | Possessive Quantifier |
|-------------------|-----------------|-----------------------|
| m^* | $m^*?$ | m^{*+} |
| m^+ | $m^+?$ | m^{++} |
| $m^?$ | $m^{??}$ | $m^{?+}$ |
| $m\{X\}$ | $m\{X\}?$ | $m\{X\}^+$ |
| $m\{X,Y\}$ | $m\{X,Y\}?$ | $m\{X,Y\}^+$ |

Let's take an example input string a1b5, and see the behavior of the greedy, lazy, and possessive quantifiers.

If we apply a regex using the greedy quantifier, \w+\d, then it will match a1b (the longest match before backtracking starts) using \w+, and 5 will be matched using \d; thus, the full match will be a1b5.

Now, if we apply a regex using the non-greedy quantifier, \w+?\d, then it will match a (the shortest match before expanding starts) using \w+?, and then the adjacent digit 1 will be matched using \d. Thus, the first full match will be a1. If we let the regex execute again, then it will find another match, b5.

Finally, if we apply a regex using the possessive quantifier, \w++\d, then it will match all the characters a1b5 (the longest possible match without giving back) using \w++. Due to this, \d remains unmatched, and hence the regex fails to find any match.

Let's take another example. The requirement is to match a string that starts with lowercase English alphabets or hyphen. The string can have any character after the alphabets/hyphens, except a colon. There can be any number of any characters of any length after the colon until the end.

An example of a valid input is `as-df999` and that of an invalid input is `asdf-:123`.

Now, let's try solving this regex problem using a greedy quantifier regex:

`| ^[a-z-]+[^:]*$`

Unfortunately, this is not the right regex pattern because this regex will match both the aforementioned valid and invalid inputs. This is because of the backtracking behavior of the regex engine in greedy quantifiers. The `[a-z-]+` pattern will find the longest possible match in the form of `asdf-`, but due to the negated character class pattern `[^:]`, the regex engine will backtrack one position to `asdf` and will match the next *hyphen* for `[^:]`. All the remaining text, that is, `:123`, will be matched using `.*`.

An example of a valid input is `as-df999` and that of an invalid input is `asdf-:123`.

Let's try to solve this regex problem using the following possessive quantifier regex:

```
| ^[a-zA-Z-]++[^\:] .*$
```

This regex pattern will still match our valid input, but it will fail to match an invalid input because there is no backtracking in possessive quantifiers; hence, the regex engine will not go back any position after matching `asdf-` in the second example string. Since the next character is a colon and our regex sub-pattern is `[^\:]`, the regex engine will stop matching and correctly declare our invalid input a failed match.

Possessive quantifiers are good for the performance of the underlying regex engine because the engine does not have to keep any backtracking information in memory. The performance increase is even more when a regex fails to match because possessive quantifiers fail faster.

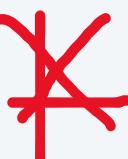
Another example of describing a pattern with a generalized RE

An e-mail address is

- A sequence of letters, followed by
- the character "@", followed by
- the character ".", followed by a nonempty sequence of lowercase letters, followed by
- [any number of occurrences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Q. Give a generalized RE for e-mail addresses.

A. $[a-z]^+@([a-z]^+\.)^+(edu|com)$

Exercise. Extend to handle rs123@princeton.edu, more suffixes such as .org, and any other extensions you can think of.

Next. Determining whether a given string matches a given RE.

Regular Expressions with Java

Week 14 – Presentation 4

REs in Java

Java's String class implements GREP.

```
public class String  
...  
boolean matches(String re) does this string match the given RE?  
...
```

```
String re = "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H";  
String zincFinger = "CAASCGGPYACGGWAGYHAGAH";  
boolean test = zincFinger.matches(re);
```

true!



Java RE client example: Validation

```
public class Validate
{
    public static void main(String[] args)
    {
        String re = args[0];
        while (!StdIn.isEmpty())
        {
            String input = StdIn.readString();
            StdOut.println(input.matches(re));
        }
    }
}
```

Applications

- Scientific research.
- Compilers and interpreters.
- Internet commerce.
- ...

Does a given string match a given RE?

- Take RE from command line.
- Take strings from StdIn.

need quotes to
"escape" the shell

```
% java Validate "C.{2,4}C...[LIVMFYWC].{8}H.{3,5}H"
CAASCGGPYACGGAAGYHAGAH
true
CAASCGGPYACGGAAGYHGAH
false
```

C₂H₂ type zinc finger domain

```
% java Validate "[$_A-Za-z][$_A-Za-z0-9]*"
ident123
true
123ident
false
```

legal Java identifier

```
% java Validate "[a-z]+@[a-z]+\.(edu|com)"
wayne@cs.princeton.edu
true
eve@airport
false
```

valid email address (simplified)

Beyond matching

Java's `String` class contains other useful RE-related methods.

- RE search and replace
- RE delimited parsing

```
public class String
```

```
    ...  
    String replaceAll(String re, String to)  
    String[] split(String re)
```

replace all occurrences of substrings matching RE with to

split the string around matches of the given RE

Examples using the RE "`\s+`" (matches one or more whitespace characters).

Tricky notation (typical in string processing): \ signals "special character" so "\\" means "\\" and "\\s" means "\s"

Replace each sequence of at least one whitespace character with a single space.

```
String s = StdIn.readAll();  
s = s.replaceAll("\\s+", " ");
```

Create an array of the words in `StdIn` (basis for `StdIn.readAllStrings()` method)

```
String s = StdIn.readAll();  
String[] words = s.split("\\s+");
```

Java String API for regular expressions' evaluation

| Method Signature | Purpose |
|--|--|
| <code>boolean matches(String regex)</code> | Matches the given regular expression against the string that the method is invoked on and returns true/false, indicating whether the match is successful (true) or not (false). |
| <code>String replaceAll(String regex, String replacement)</code> | Replaces each substring of the subject string that matches the given regular expression with the replacement string and returns the new string with the replaced content. |
| <code>String replaceFirst(String regex, String replacement)</code> | This method does the same as the previous one with the exception that it replaces only the first substring of the subject string that matches the given regular expression with the replacement string and returns the new string with the replaced content. |
| <code>String[] split(String regex)</code> | Splits the subject string using the given regular expression into an array of substrings (example given ahead). |
| <code>String[] split(String regex, int limit)</code> | This overloaded method does the same as the previous one but there is an additional second parameter. The <code>limit</code> parameter controls the number of times regular expressions are applied for splitting. |

Way beyond matching

java.util.regex

Java's Pattern and Matcher classes give fine control over the GREP implementation.

| | | |
|-----------------------------------|--|--|
| public class Pattern | | |
| ... | | |
| static Pattern compile(String re) | <i>parse the re to construct a Pattern</i> | ← Why not a constructor? Good question. |
| Matcher matcher(String input) | <i>create a Matcher that can find substrings matching the pattern in the given input string</i> | |
| ... | | |
| public class Matcher | | |
| ... | | |
| boolean find() | <i>set internal variable match to the next substring that matches the RE in the input. If none, return false, else return true</i> | |
| String group() | <i>return match</i> | |
| String group(int k) | <i>return the kth group (identified by parens within RE) in match</i> | |
| ... | | |

[A sophisticated interface designed for pros, but very useful for everyone.]

Java pattern matcher client example: Harvester

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester
{
    public static void main(String[] args)
    {
        String re      = args[0];
        In in         = new In(args[1]);
        String input   = in.readAll();
        Pattern pattern = Pattern.compile(re);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find())
            StdOut.println(matcher.group());
    }
}
```

Harvest information from input stream

- Take RE from command line.
- Take input from file or web page.
- Print all substrings matching RE.

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcggcggcggcggcggctg
gcgctg
gcnctg
gcgcggcggcggaggcggaggcggctg
```

harvest patterns from DNA

```
% java Harvester "[a-z]+@[a-z]+\.(edu|com)" http://www.cs.princeton.edu/people/faculty
...
rs@cs.princeton.edu
...
wayne@cs.princeton.edu
...
```

harvest email addresses from web for spam campaign.
(no email addresses on that site any more)

Using regular expressions in Java

Scanner API

A scanner is a utility class used for parsing the input text and breaking the input into tokens of various types, such as Boolean, int, float, double, long, and so on. It generates tokens of various types using regular expression-based delimiters. The default delimiter is a whitespace. Using the Scanner API, we can generate tokens of all the primitive types in addition to string tokens.

The `String`, `Pattern`, and `Matcher` classes are able to parse the input and generate tokens of the `String` type only, but the `Scanner` class is very useful for checking and generating tokens of different types from the input source. The `Scanner` instance can be constructed using the `File`, `InputStream`, `Path`, `Readable`, `ReadableByteChannel`, and `String` arguments.

| Method Signature | Purpose |
|---|--|
| Scanner useDelimiter(String pattern) | Sets this scanner's delimiter regex pattern to a String regex argument. |
| Scanner useDelimiter(Pattern pattern) | <p>This method is almost the same as the previous one but gets a <code>Pattern</code> as an argument instead of a <code>String</code>. This means that we can pass a regular expression already compiled. If we are forced to use the version with the <code>String</code> argument, the scanner would compile the string to a <code>Pattern</code> object even if we have already executed that compilation in other parts of the code.</p> <p>We will discuss the <code>Pattern</code> and <code>Matcher</code> class in the next chapter.</p> |
| Pattern delimiter() | Returns the pattern being used by this scanner to match delimiters. |
| MatchResult match() | Returns the match result of the latest scan operation performed by this scanner. |

| Method Signature | Purpose |
|-----------------------------------|--|
| boolean hasNext(String pattern) | Returns <code>true</code> if the next token matches the pattern constructed from the specified string. |
| boolean hasNext(Pattern pattern) | This method is almost the same as the previous one but gets <code>Pattern</code> as an argument instead of <code>string</code> . |
| String next(String pattern) | Returns the next token if it matches the pattern constructed from the specified string. |
| String next(Pattern pattern) | This method is almost the same as the previous one but gets <code>Pattern</code> as an argument instead of <code>string</code> . |
| String findInLine(String pattern) | Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |

| Method Signature | Purpose |
|--|--|
| String findInLine(Pattern pattern) | This method is almost the same as the previous one but gets <code>Pattern</code> as an argument instead of <code>string</code> . |
| Scanner skip(String pattern) | Skips the input that matches a pattern constructed from the specified string, ignoring delimiters. |
| Scanner skip(Pattern pattern) | This method is almost the same as the previous one but gets <code>Pattern</code> as an argument instead of <code>string</code> . |
| String findWithinHorizon(String pattern, int horizon) | Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| String findWithinHorizon(Pattern pattern, int horizon) | This method is almost the same as the previous one but gets <code>Pattern</code> as an argument instead of <code>string</code> . |

Package `java.util.regex`

Java Regular Expression API - Pattern and Matcher Classes

- The `MatchResult` interface
- Using `Pattern` class
- Using `Matcher` class
- Various methods of `Pattern` and `Matcher` classes and how to use them for solving problems involving regular expressions

The MatchResult interface

`MatchResult` is an interface for representing the result of a match operation. This interface is implemented by the `Matcher` class.

| Method Name | Description |
|--------------------------------------|--|
| <code>int start()</code> | Returns the start index of the match in the input |
| <code>int start(int group)</code> | Returns the start index of the specified capturing group |
| <code>int end()</code> | Returns the offset after the last character matched |
| <code>int end(int group)</code> | Returns the offset after the last character of the subsequence captured by the given group during this match |
| <code>String group()</code> | Returns the input substring matched by the previous match |
| <code>String group(int group)</code> | Returns the input subsequence captured by the given group during the previous match operation |
| <code>int groupCount()</code> | Returns the number of capturing groups in this match result's pattern |

Let's take an example to understand this interface better.

Suppose, the input string is a web server response line from HTTP response headers:

```
| HTTP/1.1 302 Found
```

Our regex pattern to parse this line is as follows:

```
| HTTP/1\.[01] (\d+) [a-zA-Z]+
```

Note that there is only one captured group that captures integer status code.

Let's look at this code listing to understand the various methods of the `MatchResult` interface better:

```
package example.regex;

import java.util.regex.*;

public class MatchResultExample
{
    public static void main(String[] args)
    {
        final String re = "HTTP/1\\.\\.[01] (\\d+) [a-zA-Z]+";
        final String str = "HTTP/1.1 302 Found";
```

]

```
        final Pattern p = Pattern.compile(re);
        Matcher m = p.matcher(str);

        if (m.matches())
        {
            MatchResult mr = m.toMatchResult();

            // print count of capturing groups
            System.out.println("groupCount(): " + mr.groupCount());

            // print complete matched text
            System.out.println("group(): " + mr.group());

            // print start position of matched text
            System.out.println("start(): " + mr.start());

            // print end position of matched text
            System.out.println("end(): " + mr.end());

            // print 1st captured group
            System.out.println("group(1): " + mr.group(1));

            // print 1st captured group's start position
            System.out.println("start(1): " + mr.start(1));

            // print 1st captured group's end position
            System.out.println("end(1): " + mr.end(1));
        }
    }
}
```

groupCount(): 1
group(): HTTP/1.1 302 Found
start(): 0
end(): 18
group(1): 302
start(1): 9
end(1): 12

See the Docs...

Class Summary

Class

Description

Matcher

An engine that performs match operations on a **character sequence** by interpreting a **Pattern**.

Pattern

A compiled representation of a regular expression.

<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/regex/package-summary.html>

Convert Regex to Predicate

Use `Pattern.compile().asPredicate()` method to get a predicate from compiled regular expression.

This predicate can be used with lambda streams to apply it on each token in a stream.

\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b

Case Study

Matching
special
characters
using
backslash to
escape

```
1 import java.util.*;
2 import java.util.regex.*;
3
4 public class PatternQuoteExample {
5     public static void main (String[] args) {
6         String input = "Math operators: +-*/. ";
7         boolean result;
8
9         String quoted = Pattern.quote("+-*/.");
10        System.out.println(quoted);
11
12        // regex using standard escaping
13        result = input.matches(".*\\s+\\+-\\/*/\\\\.\\s+.*");
14        System.out.println(result);
15
16        // regex Using Pattern.quote around our search string
17        result = input.matches(".*\\s+" + quoted + "\\s+.*");
18        System.out.println(result);
19
20        // regex Using \Q and \E around our search string
21        result = input.matches(".*\\s+\\Q+-*/.\\E\\s+.*");
22        System.out.println(result);
23    }
24 }
```

```
\Q+-*/.\E
true
true
true
```

```
1 import java.util.*;
2 import java.util.regex.*;
3 public class PatternSplitExample {
4     public static void main (String[] args) {
5         final String input = "value1||value2||value3";
6         final Pattern p = Pattern.compile( Pattern.quote( "||" ) );
7
8         // call split and print each element from generated array
9         // using stream API
10        Arrays.stream( p.split(input) ) // p.splitAsStream(input)
11            .forEach( System.out::println );
12    }
13 }
```

```
H:\work\CS209A_19S\notes08>javac PatternSplitExample.java

H:\work\CS209A_19S\notes08>java PatternSplitExample
value1
value2
value3
```

```
1 import java.util.List;
2 import java.util.stream.*;
3 import java.util.regex.*;
4
5 public class AsPredicateExample {
6     public static void main (String[] args) {
7         final String[] monthsArr =
8             { "10", "0", "05", "09", "12", "15", "00", "-1", "100" };
9         final Pattern validMonthPattern =
10            Pattern.compile( "^(?:0?[1-9]|1[0-2])$" );
11         List<String> filteredMonths =
12             Stream.of( monthsArr )
13                 .filter( validMonthPattern.asPredicate() )
14                 .collect( Collectors.toList() );
15         System.out.println( filteredMonths );
16     }
17 }
```

```
H:\work\CS209A_19\$>javac AsPredicateExample.java
```

```
H:\work\CS209A_19\$>java AsPredicateExample
[10, 05, 09, 12]
```

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.function.Predicate;
4 import java.util.regex.Pattern;
5 import java.util.stream.Collectors;
6
7 public class RegexPredicateExample {
8     public static void main(String[] args) {
9         // Compile regex as predicate
10        Predicate<String> emailFilter =
11            Pattern.compile("^(.+)@example.com$")
12            .asPredicate();
13
14        // Input list
15        List<String> emails = Arrays.asList(
16            "alex@example.com", "bob@yahoo.com",
17            "cat@google.com", "david@example.com"
18        );
19
20        // Apply predicate filter
21        List<String> desiredEmails =
22            emails.stream()
23                .filter(emailFilter)
24                .collect(Collectors.<String>toList());
25
26        // Now perform desired operation
27        desiredEmails.forEach(System.out::println);
28    }
29}
```

Making a predicate and using stream processing supported by regex applied to filter emails by domain

alex@example.com
david@example.com

```
1 import java.util.regex.*;
2
3 public class MatcherMatchesExample {
4     public static void main (String[] args) {
5         final Pattern pattern1 = Pattern.compile( "mastering" );
6         final Pattern pattern2 = Pattern.compile( "mastering.*" );
7         final Pattern pattern3 = Pattern.compile( "regular.*" );
8
9         String input = "mastering regular expressions";
10        Matcher matcher = pattern1.matcher(input);
11        System.out.printf( "[%s] => [%s]: %s%n",
12                           input, matcher.pattern(), matcher.matches());
13
14        // update the matcher pattern with a new pattern
15        matcher.usePattern(pattern2);
16        System.out.printf( "[%s] => [%s]: %s%n",
17                           input, matcher.pattern(), matcher.matches());
18
19        // update the matcher pattern with a new pattern
20        matcher.usePattern(pattern3);
21        System.out.printf( "[%s] => [%s]: %s%n",
22                           input, matcher.pattern(), matcher.matches());
23    }
24 }
```

H:\work\CS209A_19\\$>java MatcherMatchesExample
[mastering regular expressions] => [mastering]: false
[mastering regular expressions] => [mastering.*]: true
[mastering regular expressions] => [regular.*]: false

Using matching (we set a different pattern in java regex matcher class)

Each pattern may or may not match the string given as input ("mastering regular expressions")

```

1 import java.util.regex.*;
2 public class MatcherFindExample {
3     public static void main (String[] args) {
4         final String input =
5             "some text <value1> anything <value2><value3> here";
6
7     /* Part 1 */
8     final Pattern pattern = Pattern.compile( "<([><]*)>" );
9     Matcher matcher = pattern.matcher(input);
10    while (matcher.find()) {
11        System.out.printf( "[%d] => [%s]%n",
12                           matcher.groupCount(), matcher.group(1) );
13    }
14
15    /* Part 2 */
16    // now use similar pattern but use a named group and reset the
17    // matcher
18    matcher.usePattern( Pattern.compile( "<(?<name>[><]*)>" ) );
19    matcher.reset();
20    while (matcher.find()) {
21        System.out.printf( "[%d] => [%s]%n",
22                           matcher.groupCount(), matcher.group("name"));
23    }
24}

```

```

[1] => [value1]
[1] => [value2]
[1] => [value3]
[1] => [value1]
[1] => [value2]
[1] => [value3]

```

A template to match some elements of a string delineated by angle brackets.

The substrings that are extracted/matched are able to be obtained from the matcher group (they are numbered from 1 to the number of matches).

Part 2 shows the use of a named group instead of a numbered group (easier to read).

```
1 import java.util.regex.*;
2
3 public class MatcherAppendExample {
4     public static void main (String[] args) {
5         final String input =
6             "<n1=v1 n2=v2 n3=v3> n1=v1 n2=v2 abc=123 <v=pq id=abc> v=pq";
7
8         // pattern1 to find all matches between < and >
9         final Pattern pattern = Pattern.compile( "<[^>]+>" );
10
11        // pattern1 to find each name=value pair
12        final Pattern pairPattern = Pattern.compile( "(\\w+)=(\\w+)" );
13        Matcher enclosedPairs = pattern.matcher(input);
14
15        StringBuffer sbuf = new StringBuffer();
16        // call find in a loop and call appendReplacement for each match
17        while (enclosedPairs.find()) {
18            Matcher pairMatcher = pairPattern.matcher( enclosedPairs.group() );
19            // replace name=value with value=name in each match
20            enclosedPairs.appendReplacement(
21                sbuf, pairMatcher.replaceAll( "$2=$1" )
22            );
23        }
24        // appendTail to append remaining character to buffer
25        enclosedPairs.appendTail( sbuf );
26        System.out.println( sbuf );
27    }
28}
```

H:\work\CS209A_19\\$\notes08>java MatcherAppendExample
<v1=n1 v2=n2 v3=n3> n1=v1 n2=v2 abc=123 <v=pq id=abc> v=pq

Finding multiple matches with a loop (we don't have to know in advance how many matches there will be).

```
java.net.URL url = new URL( sURL );
```

```
106  /**
107   * Initializes an input stream from a URL.
108   *
109  * @param url the URL
110 * @throws IllegalArgumentException if cannot open {@code url}
111 * @throws IllegalArgumentException if {@code url} is {@code null}
112 */
113 public In(URL url) {
114     if (url == null) throw new IllegalArgumentException("url argument is null");
115     try {
116         URLConnection site = url.openConnection();
117         InputStream is = site.getInputStream();
118         scanner = new Scanner(new BufferedInputStream(is), CHARSET_NAME);
119         scanner.useLocale(LOCALE);
120     }
121     catch (IOException ioe) {
122         throw new IllegalArgumentException("Could not open " + url, ioe);
123     }
124 }
```

Regex are widely used in network programming.