



Day 1

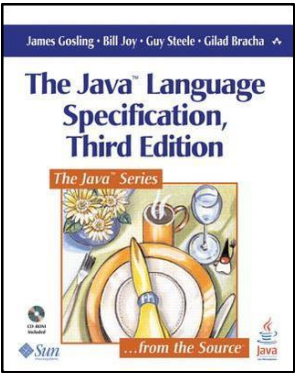


What is Java?

- Java is a platform developed by Sun Microsystems in the early 90's
- Was released in May 23 1995
- Claim to fame
 - Application are operating system independent - "Write Once, Run Everywhere"
 - Able to write program for different platforms, from microchip, mobile phone, PC, servers
- Originally Java came in 3 'flavours'
 - Java ME - Java Micro Edition
 - Java SE - Java Standard Edition
 - Java EE - Java Enterprise Edition



Java Components



Java Programming Panguage

Java Class Libraries

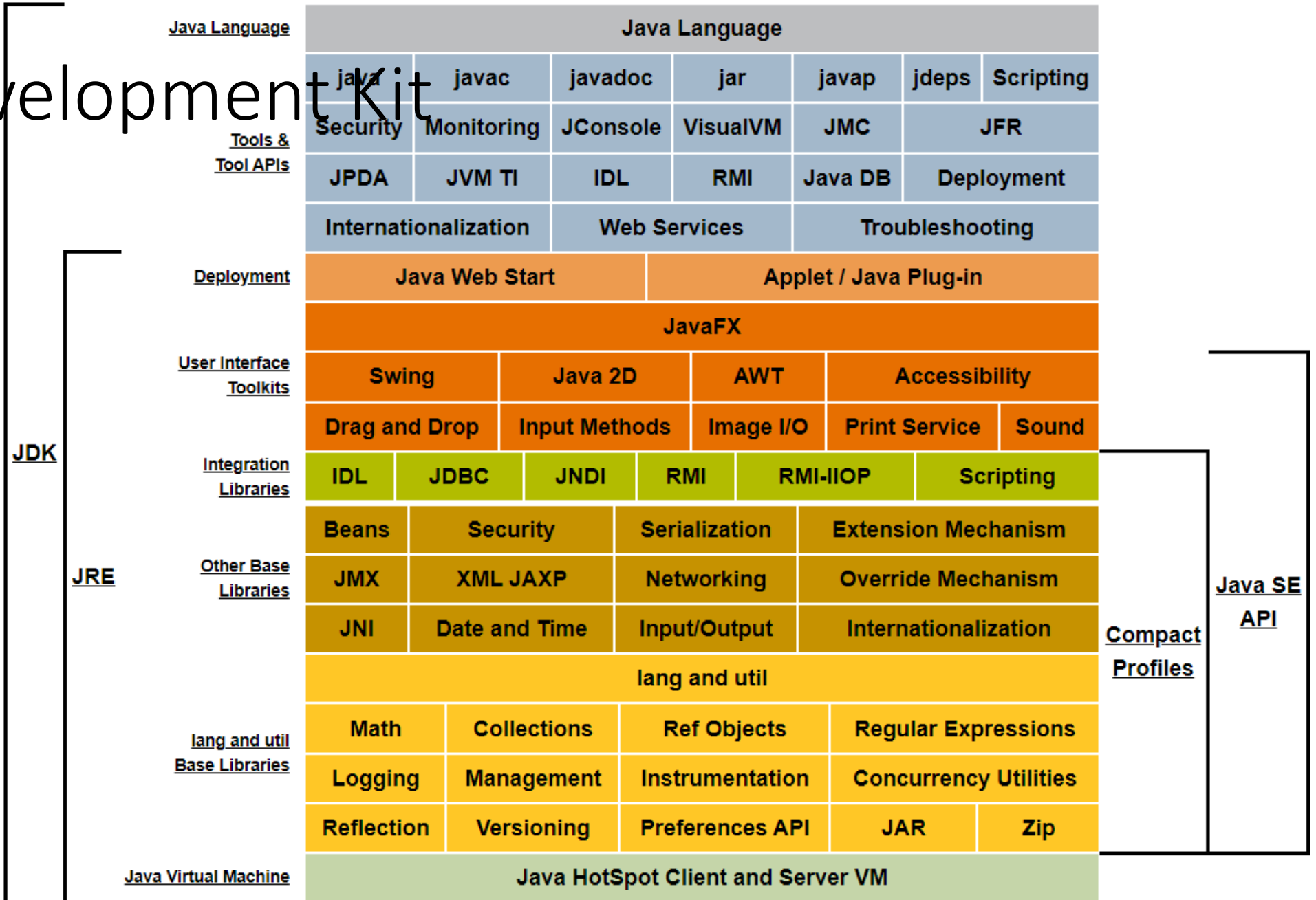
Packages	
Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.



Java Virtual Machine



Java Development Kit





Portable

Source code



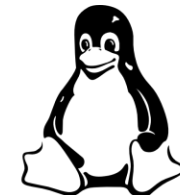
Compiler



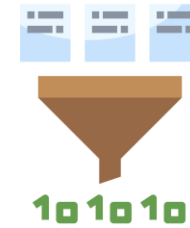
Executable



Operating System



Source code



Compiler



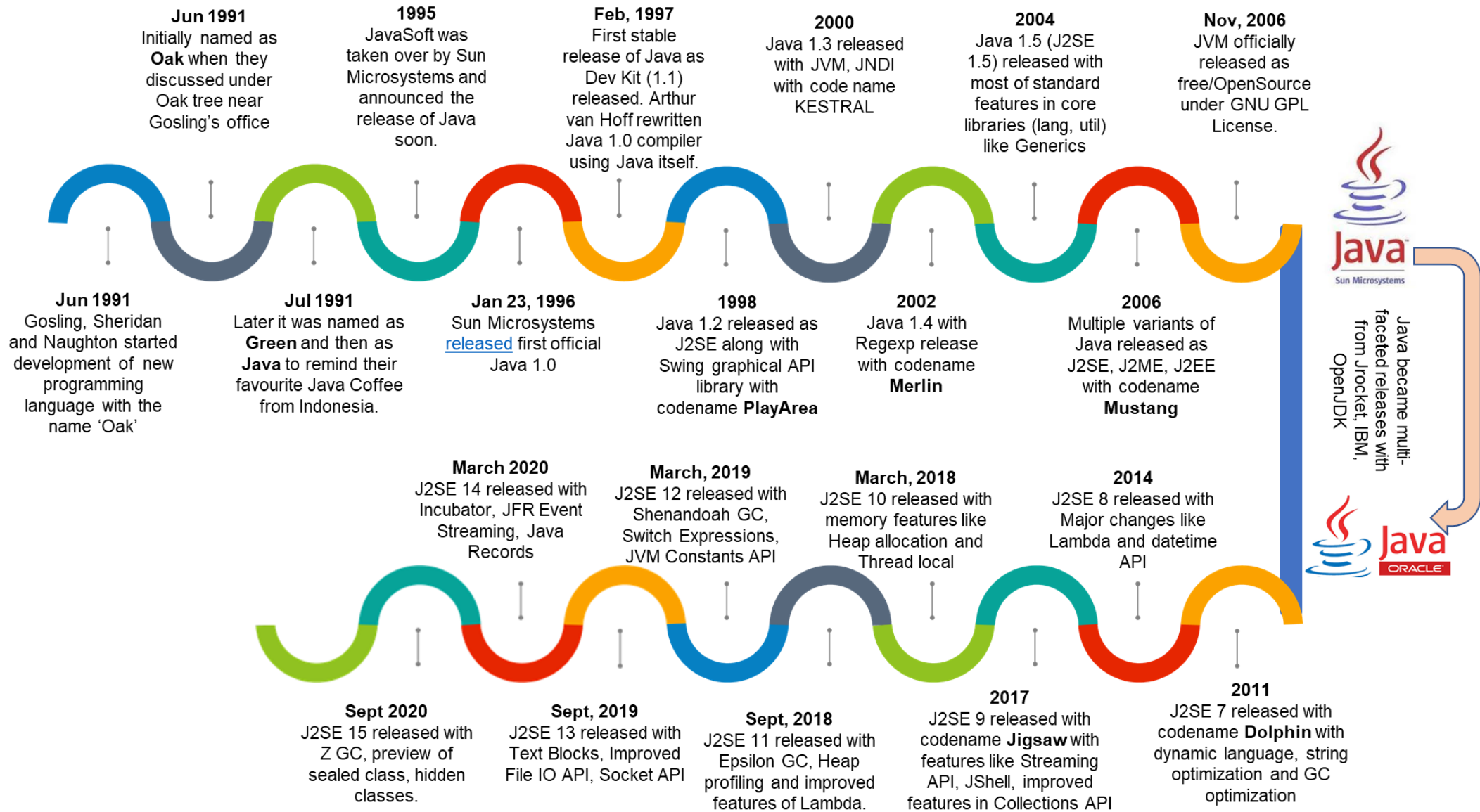
Class File



Java Virtual
Machine



Evolution of the JDK





Hello World

Directory name must be the same as package name

File name must be the same as class name
One class per file

The package name that this class belongs to

`myapp/HelloWorld.java`

`package myapp;`

List of classes from other packages

`import java.lang.*;`

Class name

`public class HelloWorld {`

`public static void main(String[] args) {`

`System.out.println("hello, world");`

Java statements are terminated with a semi-colon

System class from java.lang package

Program entry point. This is the method that the JVM will invoke to start the program

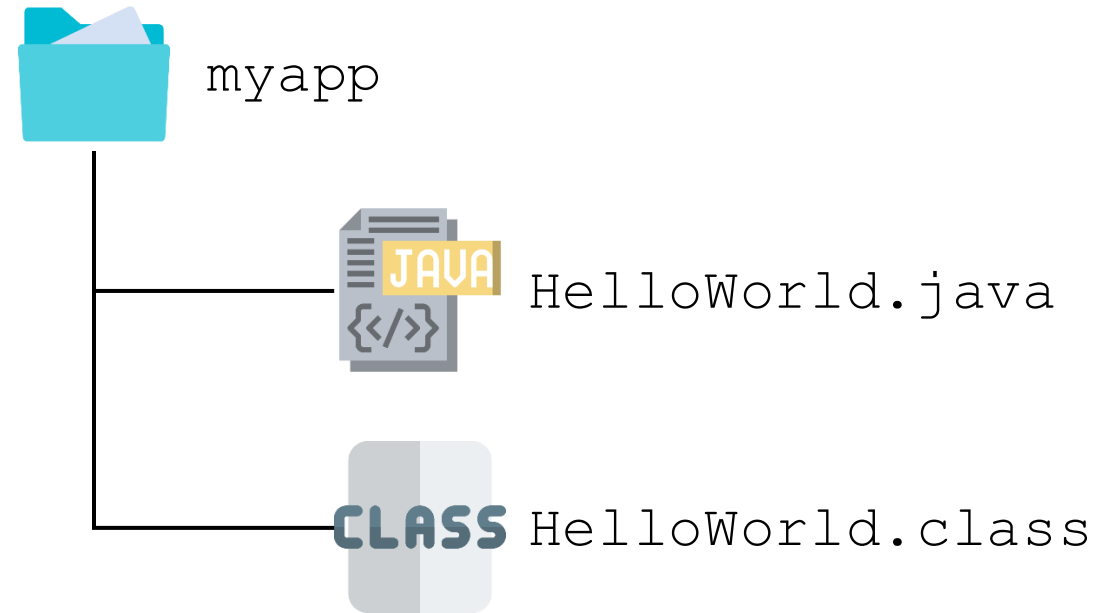
}



Compile

```
javac --source-path myapp/HelloWorld.java -d myapp
```

- `javac` is the Java compiler
- Will produce a `.class` file `HelloWorld.class` in `myapp`

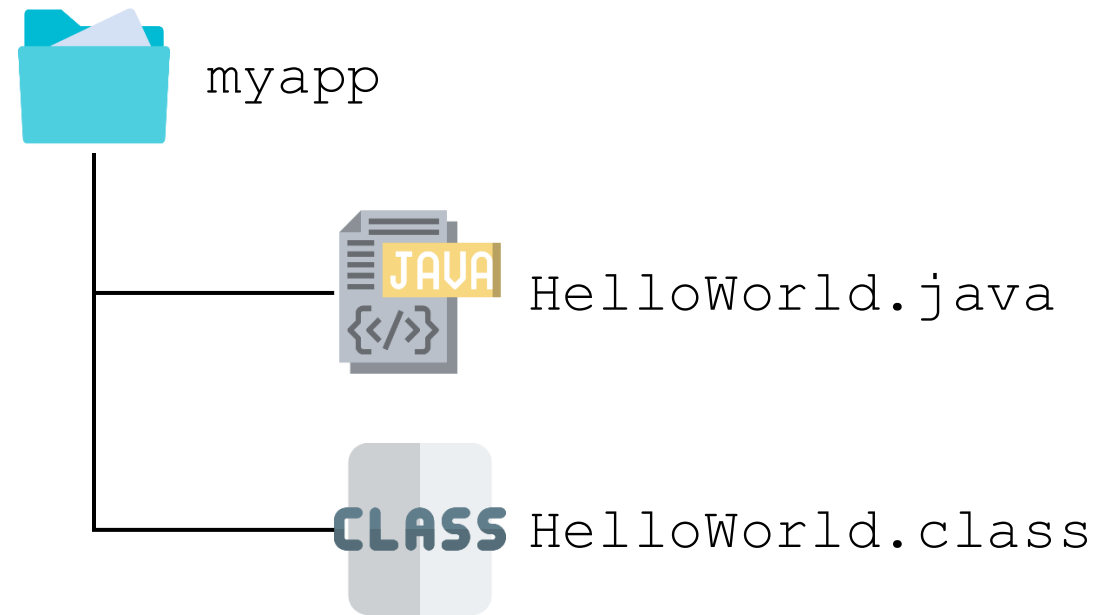




Run

```
java myapp.HelloWorld
```

- `java` starts the Java Virtual Machine (JVM)
- `myapp.HelloWorld` is the fully qualified class name
- JVM will look for a class file called `HelloWorld.class` under `myapp` directory
 - Will scan the list of directories set by the `CLASSPATH` environment variable





CLASSPATH

- Environment variable that indicate to the JVM where to look for Java classes
- One or more directories separated by
 - : (colon) - Linux / OSX
 - ; (semicolon) - Windows

```
java myapp.HelloWorld
```

```
export CLASSPATH="/opt/java/classes:."
```



CLASSPATH

```
export CLASSPATH="/opt/java/classes:./classes:."
```

```
java myapp.HelloWorld
```



- Environment variable that indicate to the JVM where to look for Java classes
- One or more directories separated by
 - `:` (colon) - Linux / OSX
 - `;` (semicolon) - Windows



Packaging Java Applications

- Java applications are packaged and delivered as a JAR (Java Archive) file
- Package all the class files into a JAR with the jar

JAR tool `jar -c -v -f myapp.jar -e myapp.HelloWorld .`

Annotations for the command:

- `jar`: JAR tool
- `-c`: Create JAR
- `-v`: Verbose
- `-f myapp.jar`: JAR file name
- `-e myapp.HelloWorld`: The main class
- `.`: JAR all files and directories in the current directory

- Execute the JAR

```
java -jar myapp.jar
```

Run the main class
in myapp.jar

- Maven, build tool can automate this process



Java Data Types

- Primitive type
 - `boolean`
 - `char` - character
 - `byte` (8), `short` (16), `int` (32), `long` (64) - integers of different sizes
 - `float` (32), `double` (64) - floating point
 - `String` - a string of character, Java treats `String` as a primitive type
- Class equivalent of primitive types, can be used interchangeably
 - `Boolean`
 - `Character`
 - `Byte`, `Short`, `Integer`, `Long`
 - `Float`, `Double`
 - `String`
 - Java automatically autobox (converts) between primitive and their equivalent classes



Example

```
import java.io.Console;
```

```
public class Main {  
    public static void main(String[] args) {
```

```
        Console cons = System.console();
```

```
        String name = cons.readLine("What is your name? ");
```

```
        System.out.printf("Hello %s. Please to meet you.\n", name);
```

```
    }
```

```
}
```

Get the console. This program
can only run from terminal

Display the prompt; read a line of
text until the new line is pressed

Formatted print

Substitute the value in name variable as
String in the position indicated by %s



Boolean Expression

Relational Operators

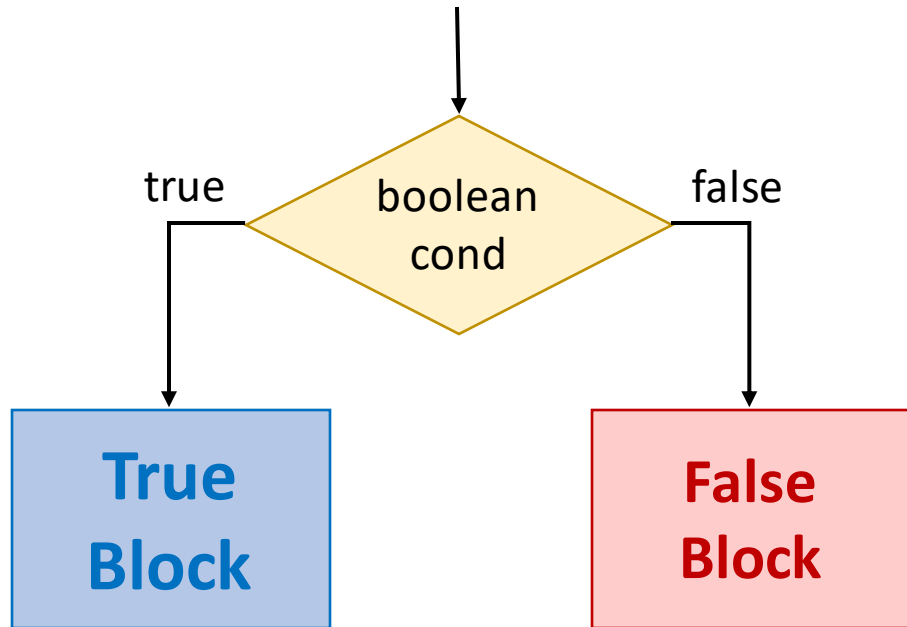
- == Equality
- != Not equals
- > Greater than
- >= Greater than or equals to
- < Less than
- <= Less than or equals to

Boolean Operators

- || Or
- && And
- ! Not



If Statement



```
String name = cons.readLine("What is your name? ");  
if (name.length() > 0) {
```

```
    System.out.printf("Hello %s\n", name);
```

```
} else {
```

```
    System.err.println(  
        "You have not told me your name");
```

```
}
```

Else is optional

All statements between the { } belongs to that branch of the condition



Example - If Statement

```
String input = cons.readLine("What is your hobby? ");
input = input.trim();
if (input.equals("swim"))
    System.out.println("The nearest public swimming pool is in Clementi");
else if (input.equals("jog"))
    System.out.println("How fast can you jog a kilometer?");
else if (input.equals("code"))
    System.out.println("Cool!");
else
    System.out.println("What is this %s hobby of yours?", input);
```

`trim()` removes the white spaces surrounding a string

String equality. Do not use `==`

Do not need `{ }` if it is just a single line



Example - Switch Statement

```
String input = cons.readLine("What is your hobby? ");
```

Where possible, good to covert string to either lower or upper case before comparison

```
switch (input.trim().toLowerCase()) {  
    case "swim": // input.equals("swim")  
        System.out.println("The nearest public swimming pool is in Clementi");  
        break;
```

```
    case "jog": // input.equals("jog")  
        System.out.println("How fast can you jog a kilometer?");  
        break;
```

```
    case "code": // input.equals("code")  
        System.out.println("Cool!");  
        break;
```

```
    default:  
        System.out.println("What is this %s hobby of yours?", input);
```

```
}
```

break exits the switch statement.
Otherwise will fallthru to the next case

Use switch when the if statement is multi branch comparison



Example - If Statement

```
String input = cons.readLine("What is your age? ");
Integer age = Integer.parseInt(input);
if (age <= 0)
    System.err.println("Are you sure?");

else if ((age > 0) && (age < 7))
    System.out.println("You are a toddler");

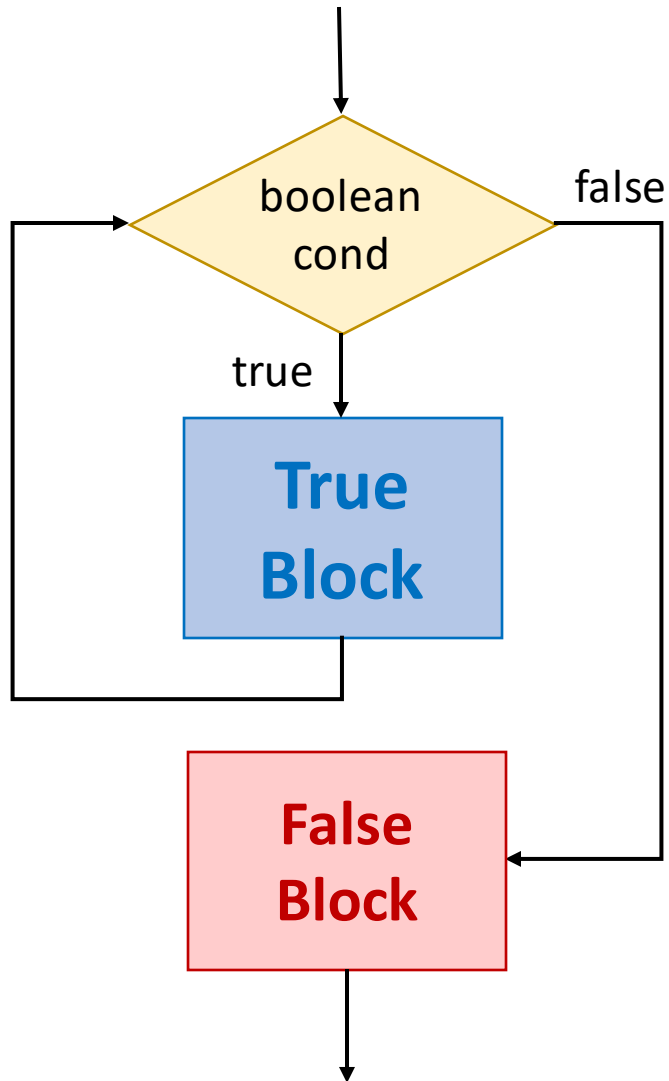
else if ((age >= 7) && (age < 12))
    System.out.println("You are a child");

else if ((age >= 12) && (age < 18))
    System.out.println("You are a teen");

else
    System.out.println("You are an adult");
```



While Loop



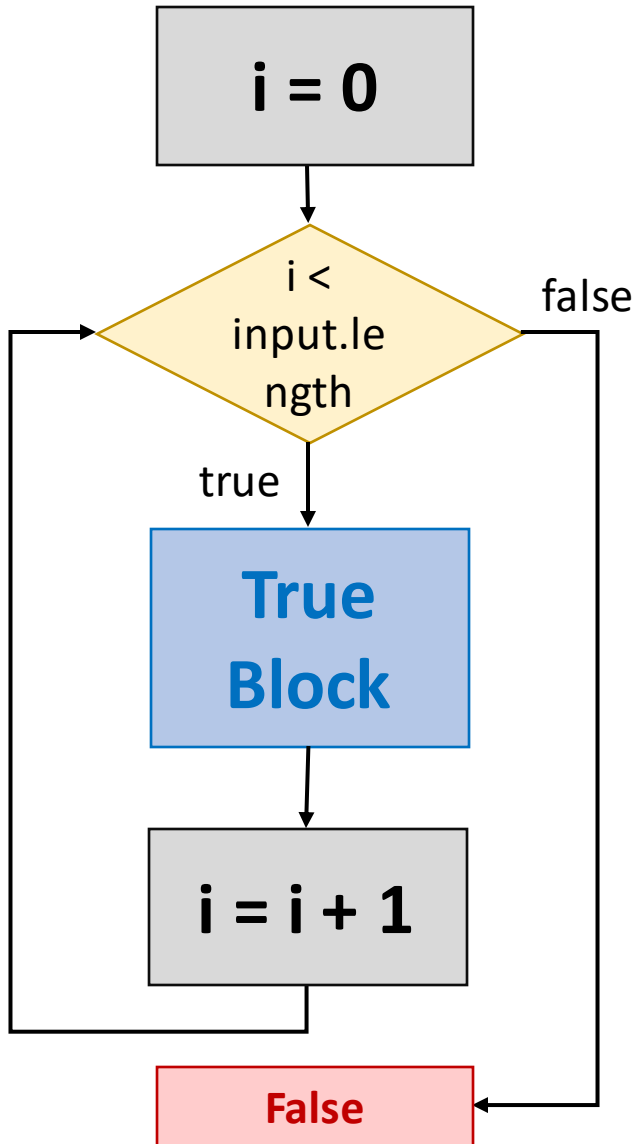
```
String name = "";
```

```
while (name.length() <= 0) {  
    name = cons.readLine("What is your name? ");  
    name = name.trim();  
}
```

```
System.out.printf("Hello %s\n", name);
```



While Loop

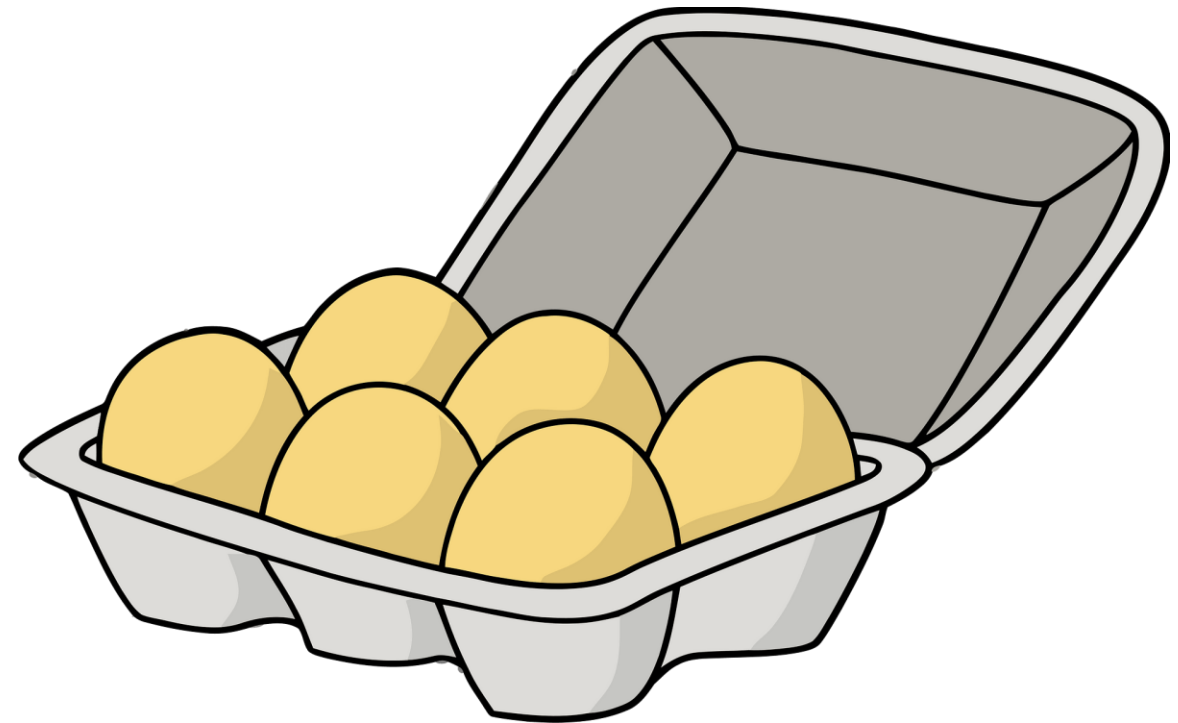


```
String input = cons.readLine("Type a phrase: ");  
for (int i = 0; i < input.length(); i++)  
    System.out.println(input.substring(0, i));
```



Array

- A variable that can store more than 1 value
- The values are indexed from 0
- Arrays have fixed size
 - Need to provide the size when the array is created
- Arrays have a length property
 - The size of the array





Example - Array

[] indicates that this is an array

Instantiate an array of type `String` with the new keyword

5 values in `todo`: `todo[0]`, `todo[1]`, .. `todo[4]`

```
Console cons = System.console();
```

```
String[] todo = new String[5];
```

length property give the size/capacity of the array

```
for (int i = 0; i < todo.length; i++) {  
    String task = cons.readLine("Enter task %d: ", (i + 1));  
    todo[i] = task;  
}
```

Assign a value into one of element in the array

```
for (int i = 0; i < todo.length; i++)  
    System.out.println(todo[i]);
```

Retrieve a value from the array indexed by `i`



Command Line Arguments

Java program entry point

Arguments are passed in via the array.
Array will be empty if no command
line arguments

```
public class HelloWorld {  
    public static void main(String[] args) {  
        String name = "fred";  
        if (args.length > 0)  
            name = args[0];  
        System.out.printf("Hello %s\n", name);  
    }  
}
```

```
java -jar hello.jar fred flintstone
```

args ["fred", "flintstone"]

```
java -jar hello.jar "fred flintstone"
```

args ["fred flintstone"]



List

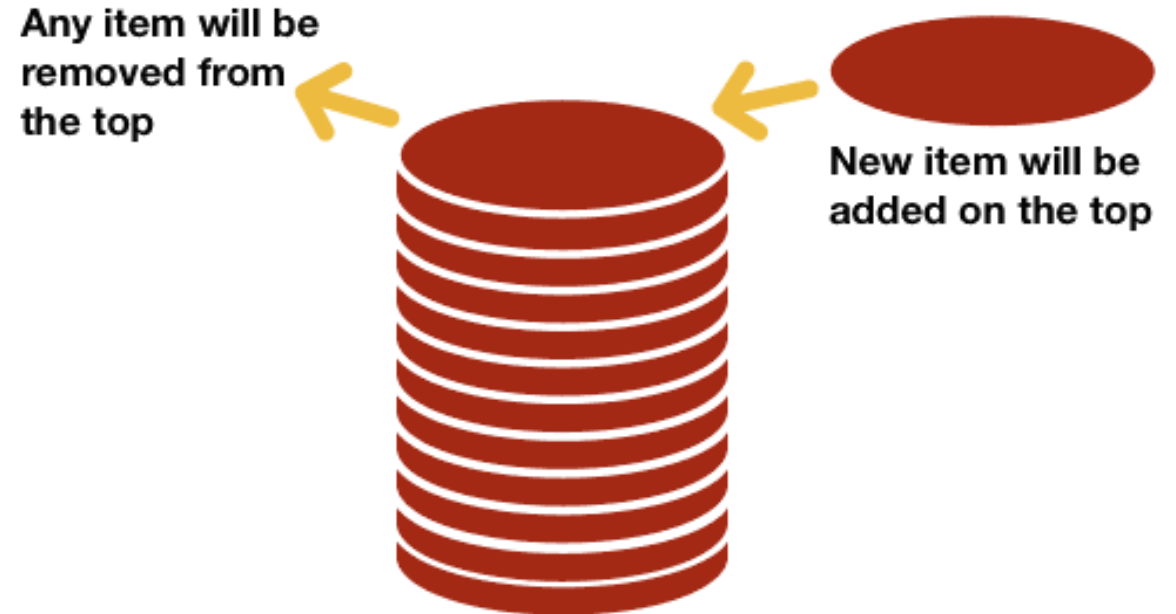


- Items can be added and removed from any where in a list
- Item will remain in the position where it is inserted
- Can add duplicate items to a list
- Items are numbered
- Some list have fixed size
- Array is a form of list



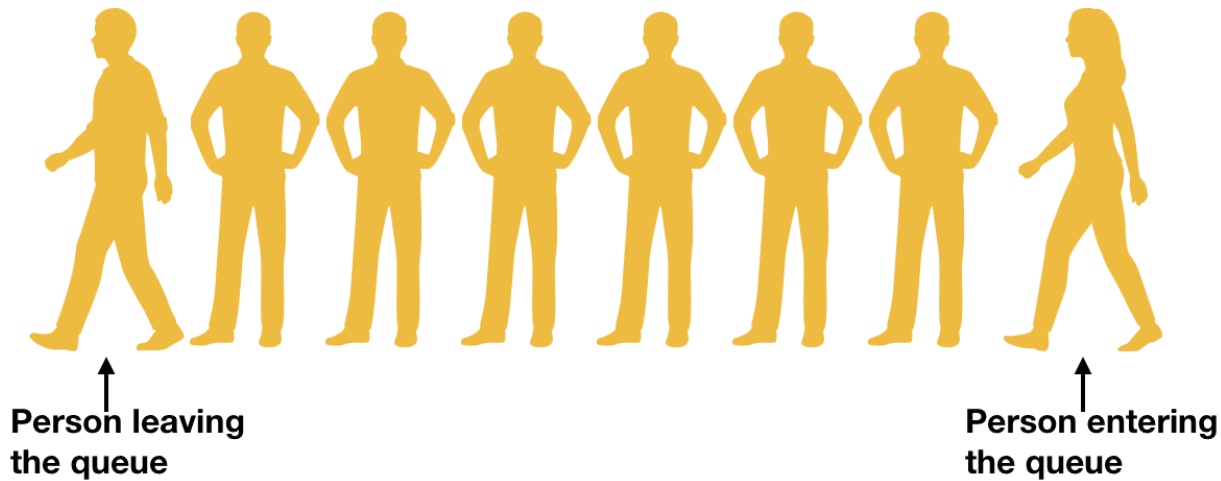
Stack

- Items can only be added to the top and removed from the top
 - Last In First Out
- Cannot access to other items from a stack except the top item
- Push - adding an item to a stack
- Pop - remove an item from a stack
- Allow duplicate





Queues



- Item are added to the back of a queue and removed from the front
 - First In First Out
- Queues preserve the arrival order of the items
- Enqueue - add an item to a queue
- Dequeue - remove an item from a queue
- Allow duplicates



Maps

- Like a list except elements are accessed with a key instead of a numerical index
- Keys have to be unique, but can have duplicate values
- Also know as dictionary

plumber /'plʌmə(r)/ *n* [C] person whose job is to fit and repair water pipes
plumbing /'plʌmɪŋ/ *n* [U] **1** system of water pipes, tanks, etc in a building **2** work of a plumber
plume /plu:m/ *n* [C] **1** cloud of sth that rises into the air **2** large feather
plummet /'plʌmɪt/ *v* [I] fall suddenly and quickly from a high level: *House prices have ~ed.*
plump /plʌmp/ *adj* having a soft, round body; slightly fat • **plump** *v* [T] ~ (up) make sth larger, softer and rounder: ~ up the pillows [PV] **plump** for sb/sth (*informal*) choose sb/sth
▶ **plumpness** *n* [U]
plunder /'plʌndə(r)/ *v* [I,T] steal things from a place, esp during a war • **plunder** *n* [U] **1** act of plundering **2** things that have been stolen, esp during a war
plunge /plʌndʒ/ *v* [I,T] (cause sb/sth to) move suddenly forwards and/or downwards: *The car ~d into the river.* ◊ *He ~d his hands into his pockets.* • **plunge** *n* [C, usu sing] sudden movement downwards or away from sth; decrease [IDM] **take the plunge** (*informal*) finally decide to do sth important or difficult ▶ **plunger** *n* [C] part of a piece of equipment that can be pushed down
pluperfect /,plʊ:'pɜ:fɪkt/ *n* (gram) = THE PAST PERFECT (PAST³)
plural /'plʊərəl/ *n* [usu sing] *adj*
◊ **p.m.** /,pi:'em/ *abbr* after 12 o'clock noon
pneumatic /nju:'mæɪtɪk/ *adj* **1** filled with air: *a ~ tyre* **2** worked by air under pressure: *a ~ drill*
▶ **pneumatically** /-kli/ *adv*
pneumonia /nju:'maʊniə/ *n* [U] serious illness affecting the lungs
PO /,pi:'əʊ/ *abbr* **1** = POST OFFICE (POST¹) **2** = POSTAL ORDER (POSTAL) ■ **P.O. box** (also '**post office box**') *n* [C] used as a kind of address, so that mail can be sent to a post office where it is kept until it is collected
poach /pəʊtʃ/ *v* **1** [T] cook fish or an egg without its shell in water that is boiling gently **2** [I,T] illegally hunt animals, birds or fish on sb else's property **3** [T] take from sb/sth dishonestly; steal sth ▶ **poacher** *n* [C] person who illegally hunts animals, birds or fish on sb else's property
◊ **pocket** /'pɒkɪt/ *n* [C] **1** small bag sewn into a piece of clothing so that you can carry things in it **2** small bag or container fastened to sth so that you can put things in it, eg in a car door or handbag **3** [usu sing] amount of money that you have to spend: *He had no intention of paying out of his own ~.* **4** small separate group or area [IDM] **in/out of pocket** (esp GB) having gained/lost money as a result of sth • **pocket** *v* [T] **1** put sth into your pocket **2** keep or take



Set



- Only store unique values, no duplicates
 - Duplicates will be removed
- Items in a set are not ordered

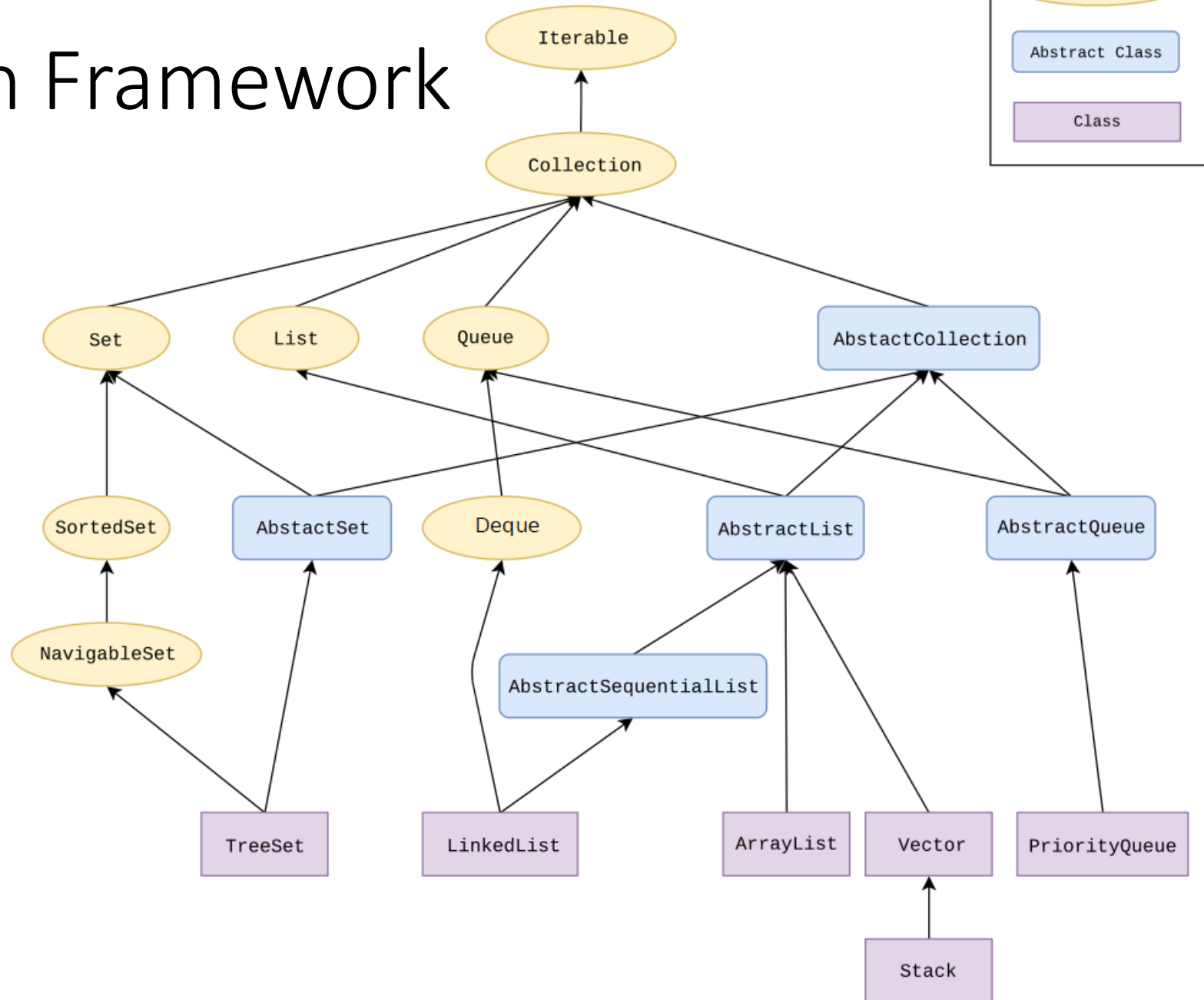


Collection Framework

- Array is a data structure for storing multiple values
 - Most programming languages support array
- Arrays are quite restrictive
 - Cannot dynamically grow or shrink
 - Have to manually implement features in set, map, stack, etc.
- Collection framework is a standard library that implements many common data structures
 - Stack, Queue, Set, List, Map
- There are many different implementation of a data structure
 - Eg List - `ArrayList`, `LinkedList`, `CopyOnWriteArrayList`
 - Select different implementation based on runtime characteristics of the list
 - Property of the data structure remains unchanged



Collection Framework



Not exhaustive



Example - List

<String> is a List of String

Instantiate a list of string.
Use the linked list
implementation of list

```
Console cons = System.console();  
List<String> myTodo = new LinkedList<>();  
String input = "";  
  
while (!"stop".equals(input)) {  
    input = cons.readLine("? ");  
  
    if (!"stop".equals(input))  
        myTodo.add(input);  
}
```

Insert into the list from the end



Example - Loop a Collection

```
for (int i = 0; i < myList.size(); i++)  
    System.out.printf("%d: %s\n", i, myList.get(i));
```

Number of elements in the list

Get the ith element. The value will be of type String because type parameter (`<String>`) of the list is String

Each iteration, item will get one element from `myList`

```
for (String item: myList)  
    System.out.printf("%s\n", item);
```

Special for loop for collections or any class that implements `Iterable`
Useful if you don't require the index



Example - Scanner

Monday go for a jog

scan.**next()** scan.**nextLine()**



```
Scanner scan = new Scanner(System.in);  
String day = scan.next();  
String description = scan.nextLine();
```

Read/scan from input





Example - Todo

Create a map where the key
and the value are `String`

```
Map<String, String> todos = new HashMap<>();
```

```
String day = "";
```

```
String todo = "";
```

```
Scanner scan = new Scanner(System.in);
```

```
while (!"stop".equals(day)) {  
    day = scan.next().toLowerCase();  
    todo = scan.nextLine().trim();
```

```
    if ("stop".equals(day))
```

Exits the loop → **break;**

```
    todos.put(day, todo);
```

Use the day as the key

```
}
```



Example - Todo

Return the keys as a Set

```
for (String day: todos.keySet())  
    System.out.printf("%s: %s\n", day, todos.get(day))
```

Use the key (day) to access the value