Java Library

- Java provides a huge library or collection of useful programs
- A gold mine of well-tested code that can save you countless hours of development time
- This huge library information is provided in API - Application Programming Interface

Random Class

- A class to create Random numbers
- Constructor Summary shows the objects of this type can be created
  - E.g. Random r = new Random();
- Method Summary shows that it can generate random values of types:
  - integers, doubles etc.
  - E.g. r.nextInt(6) – Generate a integer numbers between 0 (inclusive) and 6 (exclusive)
  - How do I generate a number between 1 and 6?

Using Library (API)

- We will use API documentation for Java Version 5
- With the help of Javadocs we can use already implemented code
- Find the documentation for the Random class
  - If you scroll down the lower left panel and click on the link labeled Random, the large "main" panel on the right will display the documentation for the Random class
Packages and import Statements

- What is a package?
  - Basically it's a directory that has a collection of related classes
  - E.g. Random Class description contains: java.util.Random
  - Indicating that the Random class code is stored in the directory path java/util/ somewhere on your machine
    - "util", or utility package

- In order to use implemented work, need to tell Java compiler where class is located
  - Use import statement
    - import java.util.Random;
    - Another way is to use the asterisk "wildcard character": import java.util.*;

Math Class

- Math Class Interface
  - Field Summary: Has two constants PI and E
  - Constructor Summary: has no public constructor
  - Methods Summary: many methods all which are static
  - Method Details: e.g. sqrt() takes a double and returns a double

Math Class Description

- Notice the phrase java.lang at the top of the main panel above the word Math
  - This means that the Math class is part of the core Java language and hence can be used directly
    - No need of an import statement

```java
public class Math{
    public static final double PI = 3.141592653589793;
    public static double sin(double d){ .. }    
    public static double sqrt(double d) { .. }
    ..
}
```

How the Math Class is Implemented

```java
public class Math{
    public static final double PI = 3.141592653589793;
    public static double sin(double d){ .. }
    public static double sqrt(double d) { .. }
    ..
}
```

```java
> Math.PI 3.141592653589793
> Math.E 2.718281828459045
> Math.sqrt(25)5.0
> Math.pow(2,10)1024
> Math.cos(0) 1.0
> Math.cos(2 * Math.PI) 1.0
```
What's different about Math Class

- It's different from OOP class
  - It is a “stateless” class
  - We only need one Math class
    - Not multiple instances
  - No need to instantiate it
    - Hence, no public constructor

- All of its variables and methods are **static**
  - static means “applies to the class as a whole” vs. “applies to an individual instance”

Dynamic Variables and Methods

- All instance variables (object data) and methods (object behavior) created without static keyword
  - Note: There is no “dynamic” keyword in Java
  - Dynamic by default

- In general, **dynamic** refers to things created at “run time” i.e. when the program is running

- Every object gets its own (dynamic) instance variables

- Every object effectively gets its own copy of each dynamic method

Example: Ticket

```java
public class Ticket{
    private static int numTicketsSold = 0; // shared
    private int ticketNum; // one per object

    public Ticket(){
        numTicketsSold = numTicketsSold + 1;
        ticketNum = numTicketsSold;
    }

    public static int getNumberSold() {
        return numTicketsSold;
    }

    public int getTicketNumber() {
        return ticketNum;
    }

    public String getInfo(){
        return "ticket # " + ticketNum + "; " +
               numTicketsSold + " ticket(s) sold."
    }
}
```

Static Variables with OO class

- **Static** means “pertaining to the class in general”, not to an individual object

- Variable is declared with the **static** keyword outside all methods
  - E.g. static int numTicketsSold;
    - There is one variable numTickets for the class *not one per object!!*

- A static variable is **shared** by all instances (if any)
  - All instances may be able read/write it
Static Methods with OO class

- A method may be declared with the `static` keyword
- Static methods live at `class level`, not at `object level`
- Static methods can `access` static variables and methods, but not dynamic ones
  - How could it choose which one? We have not created any objects yet
- Example:
  ```java
  public static int getNumSold(){
      return numTicketsSold;
  }
  ```

Static Variables & Methods in General

- A static method that is public can be accessed
  - `ClassName.methodName(args)`
- double result = Math.sqrt(25.0);
- int sold = Ticket.getNumberSold();
- boolean b = isHappy(10);
- A static variable that is public may be accessed
  - Using `ClassName.variableName`
    - E.g. Math.PI, Math.E
  - Static variables act as global variable i.e. accessible within any static method

Ticket class Interactions

```java
> Ticket.getNumberSold()
0
> Ticket t1 = new Ticket();
> t1.getTicketNum()
1
> t1.getInfo()
"ticket # 1; 1 ticket(s) sold."
> t1.getNumberSold()
1
> Ticket t2 = new Ticket();
> t2.getTicketNum()
2
> t2.getInfo()
"ticket # 2; 2 ticket(s) sold."
> t1.getInfo()
"ticket # 1; 2 ticket(s) sold."
> Ticket.getNumberSold()
2
```
**Example**

```java
public class Point {
    int x;
    int y;
    public static void main(String args[]) {
        x = 5;
        y = 10;
    }
}
```

```
all are wrong
```