

devop

intro_deployment

Deployment is the act of uploading an application to a server. Frequent deployment of an application is quite common, as each deploy may contain an updated web application with an added feature or security update. Deployment is composed of several components that work in conjunction with one another.

- **Server:** The server provides the infrastructure for communication via the web, making it possible to respond to requests from clients. In this course, the server will be using is an Amazon service called EC2.
- **Repository:** A repository host such as GitHub is commonly used among development teams. The benefits of using a shared repository extend beyond improving the development process. This empowers the team to have one workspace for testing code and another workspace for *production*, or code that is ready to be used by real users.
- **Continuous Integration:** It's tedious for a developer to manually test the code, zip the code into a file, and deploy the code. Not to mention, if the developer forgets to test one component, it could be a critical bug that renders the application useless until it's patched. Fortunately, some tools do this process of *continuous integration* on behalf of the developer. In this course, the continuous integration tool will be Travis CI.

what is cloud computing

"Cloud computing is the on-demand delivery of computing power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing." - [Amazon](#)

what is ec2

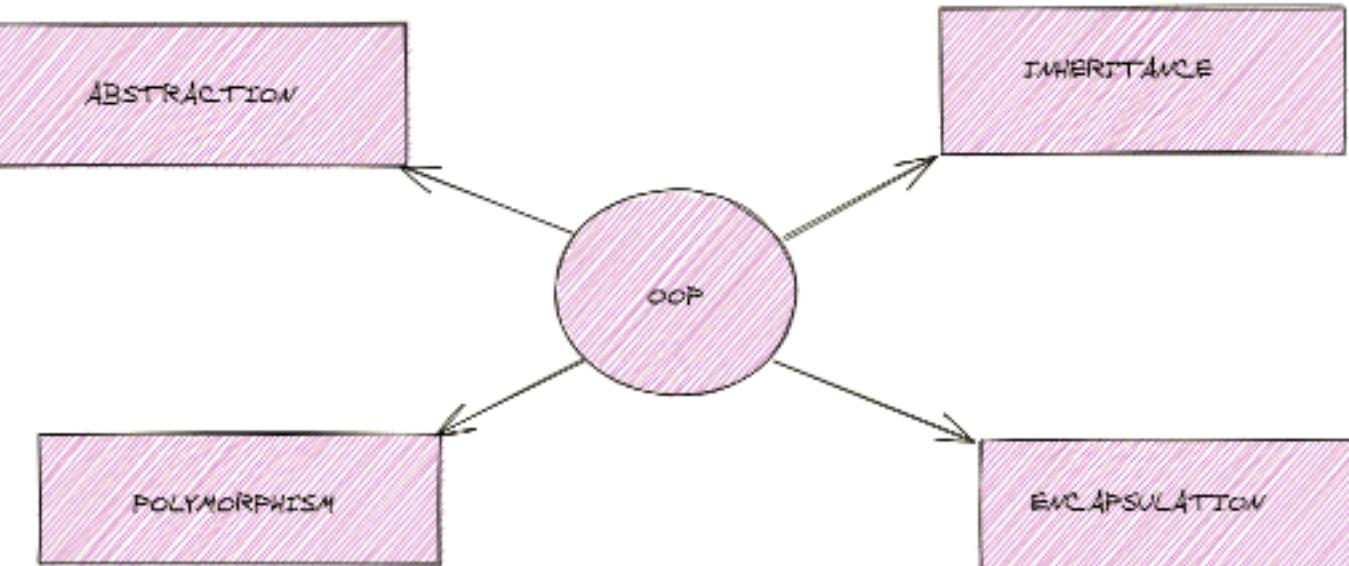
Elastic Compute Cloud (*EC2*), is Amazon's cloud computing product. There are many competitive products available for developers from which to choose, and Amazon's solution is one of the more popular options. The "elastic" in the EC2 name means that the servers are capable of *scaling*, or using more computing resources when the computing needs are high at any given time. The computing resources range from CPU speeds to data storage, which is crucial if the product surges in use during different times of the day. The pay-as-you-go pricing model translates to a lower bill when resources are less frequently used rather than paying a flat monthly premium for unused hardware.

Interview

Java is an OO language as it is modeled and organized around objects rather than actions and data rather than logic. OOP in java aims to implement real-world entities such as objects, classes, abstractions, inheritance, and so on.

real time entity with state and behavior.

4 Principle of OOP



1) Polymorphism is the ability of a variable, function, or object to take multiple forms. It allows you to define one interface or method and have multiple implementations. Polymorphism is characterized by the method overloading and method overriding.

Abstraction: Abstraction is used to hide internal details and showing only the functionality. In Java, we use abstract class and interface to achieve abstraction.

Encapsulation: Encapsulation means to bind or wrap code and data together into a single unit. A Java class is an example of Encapsulation.

Polymorphism: Polymorphism means that one task is performed in different ways. There are 2 types of Polymorphism: Run-time Polymorphism and Compile-Type polymorphism.

Inheritance: Inheritance means acquiring all the properties and behavior of an object. It provides code re-usability.

Class

A class in Java is a blueprint that includes all your data. A class contains variables and methods that describe the behavior of an object.

TEXT/X-JAVA

```
1 class Vehicle {  
2  
3     //variables  
4     private int numWheels;  
5     private String name;  
6  
7     //methods  
8     public String getName() {  
9         return name;  
10    }  
11  
12    public void setNumWheels(int numWheels) {  
13        this.numWheels = numWheels;  
14    }  
15  
16  
17 }
```

collection of similar Objects.

Objects

An object is an instance of a class that can access your data. The keyword `new` is used to create the object.

TEXT/X-JAVA

```
1 //Declaring and initializing an object  
2 Vehicle v = new Vehicle();
```

Constructors

A constructor is a block of code that initializes a newly created object. It is basically a method but doesn't have any return type and has the same name as its class. There are two types of constructors in Java:

1) Default Constructor

The default constructor is created by default by the java compiler at class creation if no other constructor is declared in the class. It doesn't contain any parameters which is why it is sometimes referred to as the no-argument constructor.

TEXT/X-JAVA

```
1 public class Vehicle {  
2  
3     //default constructor  
4     public Vehicle() {}  
5  
6 }  
7  
8 public class Driver {  
9  
10    public static void main(String[] args) {  
11  
12        //creating vehicle object using default constructor  
13        Vehicle vehicle = new Vehicle();  
14    }  
15  
16 }
```

Constructor : special function used to initialize state of an object. No return Type but returns current instance of the class. Not inherited so cant make final. Called by super() method from child class. Cant have this() and Super() together in constructor as both should be the first statement.

Overloaded

The overloaded constructor contains one or more parameters. It is used to provide different values to the distinct objects at the time of their creation.

```
public class Vehicle {  
    int numWheels;  
    String color;  
    int numDoors;  
  
    //overloaded constructor  
    public Vehicle(int numWheels, String color, int numDoors) {  
  
        //keyword this points to current object  
  
        this.numWheels = numWheels;  
        this.color = color;  
        this.numDoors = numDoors;  
    }  
  
    //toString() method used get value of instance variables  
    public String toString() {  
  
        String s = "number of wheels" + numWheels +  
            "color:" + color + "number of doors:" + numDoors;  
  
        return s;  
    }  
}  
public class Driver {  
  
    public static void main(String[] args) {  
  
        Vehicle car = new Vehicle(4, "red", 4);  
        Vehicle truck = new Vehicle(12, "black", 2);  
  
        System.out.println(car.toString());  
        System.out.println(truck.toString());  
    }  
}
```

info

A constructor is defined as a member function in a class which has the same name as of class.

It is automatically called at the run-time whenever an object class is created. It doesn't have any return type.

Need of Constructor: There might be a situation where exposing the class variable to the main program is not secured. At that time the class variables can be declared as private because the private variables are not accessible from the other class. In such a situation, if constructors are defined, then the main method need not to access the class variables directly.

Default

Default Constructor: In Java, the Default constructor is the one which does not accept any input. It is used to initialize the instance variable with the default values and is majorly used for object creation. A default constructor is invoked implicitly by the compiler if no constructor is defined by the user.

Parameterized

Parameterized Constructor: In Java, the Parameterized constructor is the one which is capable of initializing the instance variables with the given values. In other words, those constructors that accept the arguments are called parameterized constructors.

Example answer: "A constructor initializes a *newly created object*. Java supports *copy constructor*, but you *have to write your own code to do it*. *Constructor chaining* is calling a constructor from another constructor. However, you *cannot call a sub-class constructor using a super-class constructor*."

Method Overloading

Method Overloading is the act of having the same function name but differs in number and type of arguments within the same class.

TEXT/X-JAVA

```
1 public class Example {  
2  
3     public int sum(int a, int b) {  
4         return a + b;  
5     }  
6  
7     public int add(int a, int b, int c) {  
8         return a + b + c;  
9     }  
10    }  
11  
12  
13 }
```

Method Overloading is a feature that allows a class to have more than one method with the same name and different parameters.

We can overload the method by either changing number of arguments or changing the data

```
public class Test {  
  
    public static int add(int a, int b) {return a+b};  
  
    public static int add(int a, int b, int c) {return a+b+c};  
  
}  
  
public class TestOverloadingMethod {  
  
    public static void main(String[] args)  
  
    { System.out.println(Test.add(10,11));  
  
        System.out.println(Test.add(10,11,12)); }  
}
```

Method Overriding is a feature that allows a sub-class(child class) to have the same method (with the same name and same parameters) as that in the superclass(parent class).

Method Overriding is used for run time polymorphism

```
public class Vehicle
{
void run()
{
System.out.println("Vehicle is running");
}
}

//Creating a child class
class Bike extends Vehicle{
public static void main(String args[]){
//creating an instance of child class
Bike obj = new Bike();
//calling the method with child class instance
obj.run();
}
}
```

Overriding

Method Overriding is the specific implementation of a method in a child class which is already defined in the parent class.

TEXT/X-JAVA

```
1 public class Vehicle {  
2  
3     //Overridden method  
4  
5     public void drive() {  
6  
7         System.out.println("Vehicle is driving");  
8  
9     }  
10 }  
11  
12 public class Car extends Vehicle {  
13  
14     //Overridden Method  
15  
16     public void drive() {  
17  
18         System.out.println("Car is driving");  
19  
20     }  
21  
22 }
```

As you can see, the child class `Car` inherited the `drive` method from its parent class `Vehicle`. However, we can change the functionality of the method while keeping the same method signature.

2) Abstraction is a concept of hiding the details and showing only the necessary things to the user. We use an abstract class/Interface to express the intent of the class rather than the actual implementation.

Overload main

Yes, we can overload `main()` method like other methods.

But when we run the program, the program doesn't execute the overloaded `main` method, so we need to call the overloaded `main` method from the actual `main` method only.

Method

```
import java.io.*;
import java.util.*;

class Sports{
    String getName(){
        return "Generic Sports ";
    }
    void getNumberOfTeamMembers(){
        System.out.println( "Each team has n players in " + getName() )
    }
}
```

```
class Soccer extends Sports{
    @Override
    String getName(){
        return "Soccer Class ";
    }
    @Override
    void getNumberOfTeamMembers(){
        System.out.println( "Each team has 11 players in " + getName() )
    }
}
```

```
public class Solution {

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output
        to STDOUT. Your class should be named Solution. */
        Sports sports = new Sports();
        Soccer soccer = new Soccer();
        System.out.println(sports.getName());
        sports.getNumberOfTeamMembers();
        System.out.println(soccer.getName());
        soccer.getNumberOfTeamMembers();
    }
}
```

Super Keyword

```
import java.io.*;
import java.util.*;

class BiCycle{
    String define_me(){
        return "a vehicle with pedals.";
    }
}

class MotorCycle extends BiCycle{
    String define_me(){
        return "a cycle with an engine.";
    }
}
```

```
MotorCycle(){
    System.out.println("Hello I am a motorcycle, I am "+ define_me());
}

String temp = super.define_me();

System.out.println("My ancestor is a cycle who is "+ temp );
}

}

class Solution{
    public static void main(String []args){
        MotorCycle M = new MotorCycle();
    }
}
```

Abstract

An Abstract class is a class which is declared with an abstract keyword and cannot be instantiated.

TEXT/X-JAVA

```
1 public abstract class Example{  
2  
3     public void display();  
4  
5     public int sum(int a, int b);  
6  
7 }
```

Abstract

```
import java.io.*;
import java.util.*;

abstract class Book{
    String title;
    abstract void setTitle(String s);
    String getTitle(){
        return title;
    }
}

class MyBook extends Book {

    @Override
    void setTitle(String s){
        this.title = s;
    }
}
```

```
public class Solution {
    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output
        to STDOUT. Your class should be named Solution. */
        Scanner sc=new Scanner(System.in);

        String title = sc.nextLine();

        MyBook new_novel = new MyBook();

        new_novel.setTitle(title);

        System.out.println("The title is: "+new_novel.getTitle());
    }
}
```

Interface

An interface is a blueprint that contains static constants and abstract methods.

TEXT/X-JAVA

```
1 public Interface VehicleInterface {  
2  
3     //methods in an interface does not have a body  
4     public void displayVehicle();  
5     public boolean isAutomatic();  
6  
7 }  
8  
9 public class Vehicle implements VehicleInterface {  
10  
11     public void displayVehicle() {  
12  
13         //The body of displayVehicle goes here  
14  
15         System.out.println("This is a car");  
16  
17     }  
18  
19     public boolean isAutomatic() {  
20  
21 }
```

Interface

```
import java.util.*;  
  
interface AdvancedArithmetic{  
    int divisor_sum(int n);  
}  
  
class MyCalculator implements AdvancedArithmetic{  
    int sum = 0;  
  
    public int divisor_sum(int n){  
        for(int i=1;i<=(n+1)/2;i++){  
            if(n%i==0){  
                sum+=i;  
            }  
        }  
        if(n==1){  
            return 1;  
        }  
        return (sum+n);  
    }  
}
```

```
class Solution{
    public static void main(String []args){
        MyCalculator my_calculator = new MyCalculator();
        System.out.print("I implemented: ");
        ImplementedInterfaceNames(my_calculator);
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.print(my_calculator.divisor_sum(n) + "\n");
        sc.close();
    }

    static void ImplementedInterfaceNames(Object o){
        Class[] theInterfaces = o.getClass().getInterfaces();
        for (int i = 0; i < theInterfaces.length; i++){
            String interfaceName = theInterfaces[i].getName();
            System.out.println(interfaceName);
        }
    }
}
```

Inheritance

Inheritance is a concept where the properties of one class can be inherited by the other. It helps with reusability and establishes a relationship between different classes. The relationship is established in code using the keyword `extends`.

```
public class Person {  
  
    private int pID;  
    private String name;  
  
    Person() {  
        pID = 4;  
        name = "Brian";  
    }  
  
    Person(int pID, String name) {  
  
        this.pID = pID;  
        this.name = name;  
    }  
}
```

```
public class Student extends Person {  
  
    public void displayName() {  
  
        System.out.println(super.name);  
    }  
  
}
```

In this example, the child class `Student` inherits attributes from its parent class, `Person`. As stated above the relationship is established using the keyword `extends`. The `super` keyword in java acts as a reference variable that is used to refer to parent class objects.

The `displayName()` method would print the value of `name` that is inherited from the `Person` class.

Inheritance 1

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

class Animal{
    void walk(){
        System.out.println("I am walking");
    }
}

class Bird extends Animal implements asd{
    void fly(){
        System.out.println("I am flying");
    }
    public void sing() {
        System.out.println("I am singing");
    }
}
```

```
interface asd{
    void sing();
}

public class Solution{

    public static void main(String args[]){

        Bird bird = new Bird();
        bird.walk();
        bird.fly();
        bird.sing();

    }
}
```

Inheritance ii

```
import java.io.*;
import java.util.*;

class Arithmetic {
    Arithmetic(){}
}

class Adder extends Arithmetic {
    Adder(){}
    int add(int a, int b) {
        return a + b;
    }
}
```

```
public class Solution{
    public static void main(String []args){
        Adder a = new Adder();

        System.out.println("My superclass is: " + a.getClass()
        .getSuperclass().getName());

        System.out.print(a.add(10,32) + " " + a.add(10,3) + " " + a.add
        (10,10) + "\n");
    }
}
```

Encapsulation

Encapsulation is a process of hiding data implementation by restricting access to public methods. The instance variables are kept private and the accessor methods(getters and setters) are made public.

wrapping data and associated function into single unit implements data hiding using private property accessed using getter and setter methods.

Static

Static Variable : belong to class and get memory only once in class area at the time of class loading.

Static Method : belong to class, cant use non static variables and methods inside if it is not known

Static Block : initialize static variables and executed before main method at the time of class loading.

Static Import : access any static member of a class directly. There is no need to qualify it by the class name in program.

Keyword

8. Static keyword in Java. The **static** keyword is a very popular question in the interviews.

What's a static keyword in Java?

- *Static variables and methods belong to the class, not to a specific object. We need to use static members by class name.*

```
public class Person {  
    public String name;  
    public int age;  
    public static String address;  
  
    public static void main(String[] args){  
        Person john = new Person();  
        john.name = "John";  
        john.age = 35;  
        john.address = "101 Main St";  
  
        System.out.println(john.name);  
        System.out.println(john.age);  
        System.out.println(john.address);  
  
        Person smith = new Person();  
        System.out.println(smith.name);  
        System.out.println(smith.age);  
        System.out.println(smith.address);  
    }  
}
```

```
/*
```

Output:

John

35

101 Main St

null

0

101 Main St

- Static variables belong to class. They do not belong to specific object.

That's why for second object print "101 Main St" value for address.

Correct way of accessing static members is by class name

```
*/
```

Static Initializer Block

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {
    static boolean flag = true;
    static int B,H;
    static{
        Scanner scan = new Scanner(System.in);

        B = scan.nextInt();

        scan.nextLine();

        H = scan.nextInt();
        scan.close();
    }
}
```

```
if((B > 0) && (H > 0)){
    flag = true;

}else if((B <=0 ) || (H <= 0)){
    flag = false;

    System.out.println("java.lang.Exception: Breadth and
height must be positive");
}

}

public static void main(String[] args){
    if(flag){
        int area = B * H;
        System.out.print(area);
    }
}

}//end of main

}//end of class
```

Access Modifiers

Public > Protected > Default > Private

Public: Public class, public methods, or public variables can be accessed from anywhere.

Protected: Protected methods or fields can be accessed from the same class, or subclass or from the class within the same package.

Private: Private methods or fields can be accessed only from the same class to which they belong.

Default: Default methods or Default fields or Default class can be accessed only from the same class or from the same package.

Final

In Java, Final keyword can be used with variable, method and class.

If a variable is declared using Final keyword, then its value can be assigned only once and after assignment, it can't be changed.

If a method is declared using Final keyword, then this method cannot be overridden by the subclasses.

If a class is declared using Final keyword, then this class cannot be inherited.

variable (can't change, constant), method(can't override), class (can't inherit)

Package

group of similar classes, interface and sub package.

java.lang package is imported implicitly(Throwable, Iterable, Comparable, Object).

this

this : points current object, it is final type, can be used in synchronized block

Difference between this and supper

this()

It represents the current instance of a class.

It is used to call the default constructor of the same class.

It is used for pointing the instance of a current class.

It is used to access methods of the current class.

super()

It represents the current instance of parent class.

It is used to call the default constructor of the parent class.

It is used to point the instance of parent class.

It is used to access the methods of parent class.

Difference Between an inner class and a sub-class

An inner class is defined as a class which is nested within another class. An inner class can access all methods and variables that are defined in the outer class.

Sub-class is defined as a class which is inherited from some another class which is called a superclass. A subclass can access only public and protected method and fields of its superclass.

Difference between an abstract and interface class

Abstract Class	Interface
It can have abstract and non-abstract methods	It can have abstract methods only
Abstract keyword is used to declare an abstract class	Interface keyword is used to declare an interface.
It can extend another Java class and implement multiple Java interfaces.	It can extend another Java interface only
It can have class members like private, protected, etc.	Members of Interface are public by default
It can have final, non-final, static and non-static variables	It has only static and final variables.
It can provide the implementation of interface	It can't provide the implementation of abstract class

Local variables and Instance variables

Local Variables: Local Variables are defined within a block or method or constructor.

These variables are created when the block is entered and destroyed after exiting from the block. We can access these variables only within the block where they are declared. We cannot use any access specifier with local variables.

Instance Variables: Instance variables are declared in a class outside any method, block or constructor.

These variables are created when an object of the class is created and destroyed when the object is destroyed. We can access these variables only by creating objects.

We can use access specifiers for instance variables. If no access specifier is specified then the default access specifier is used.

Main Method Private

The main method must always be public static in order to run any application correctly in Java. If the main method is declared as private, then it will not give any compilation error but the program will not get executed and will give a run-time error.

Thread

Thread is a piece of code which can be executed independently.

In Java, all program has at least one thread called as the main thread that is created by JVM. The main thread is used to invoke the main() method of program. Threads are executed concurrently.

various ways to create a Thread

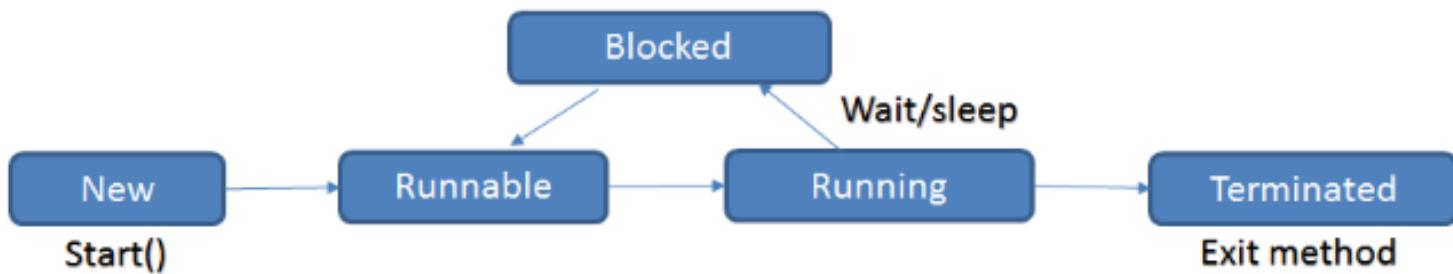
- By extending **Thread class**:

```
public class Test extends Thread{public void test2() {}}
```

- By **Implementing Runnable Interface**:

```
public class Test implements Runnable{ public void test2() {}}
```

15. Explain Thread life cycle in Java.



New: In this state, the thread has been created but start() method is not invoked yet.

Runnable: The thread is in the runnable state after start() method is invoked but before the invocation of run() method.

Running: The thread is in running state after the invocation of run() method.

Blocked: The thread is alive but it is not eligible to run.

Terminated: The thread is in the terminated state when the run() method is completed

Collections

A collection is a group of objects represented as a single unit. It is a framework that provides an architecture to store the objects and manipulate a group of objects.

Collections are used to perform the following operations:

- Searching
- Sorting
- Manipulation
- Insertion
- Deletion

Exception

Exception is a problem that occurs when the normal flow of program is disrupted and therefore program or application terminates abnormally.

Exceptions are a subclass of `java.lang.Exception`

To maintain the normal flow of application we can handle the Exception through Exception handling.

Exception handling is a process to handle run-time errors such as `IOException`, `SQLException`, etc.

type

Checked Exceptions: Checked Exceptions are the exceptions that are checked at the time of compilation by compiler. These type of exceptions should be handled by the programmer.

Eg.: FileNotFoundException, SQLException,etc.

Unchecked Exceptions: Unchecked Exceptions are the exceptions that are not checked at the time of compilation but they occur at the time of execution. They are also called as Runtime Exceptions. Eg.:
ArithmaticException, NullPointerException.

Try Block

Try block always needs to be followed by either catch block or finally block or both.

If any exception is thrown from try block, then it should either be caught in catch block or if any specific tasks needs to be performed before abortion, then they are put in a finally block.

```
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN. Print output
to STDOUT. Your class should be named Solution. */
    Scanner scan = new Scanner(System.in);
    try{
        int x = scan.nextInt();
        int y = scan.nextInt();
        System.out.println(x/y);

    } catch(Exception e){
        e.printStackTrace();
        if(e instanceof java.lang.ArithmaticException){
            System.out.println(e.toString());
        }
        if(e instanceof java.util.InputMismatchException){
            System.out.println("java.util.InputMismatchException");
        }
    }
}
```

Try Catch

The try statement **allows you to define a block of code to be tested for errors while it is being executed**. The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

Exception Handling

```
class MyCalculator {  
    public static int power(int n, int p) throws Exception{  
        if(n < 0 || p < 0){  
            throw new Exception ("n or p should not be negative.");  
        }else if(n==0 && p ==0){  
            throw new Exception("n and p should not be zero.");  
        }else{  
            return ((int)Math.pow(n,p));  
        }  
    }  
}
```

```
public class Solution {  
    public static final MyCalculator my_calculator = new MyCalculator();  
    public static final Scanner in = new Scanner(System.in);  
  
    public static void main(String[] args) {  
        while (in.hasNextInt()) {  
            int n = in.nextInt();  
            int p = in.nextInt();  
  
            try {  
                System.out.println(my_calculator.power(n, p));  
            } catch (Exception e) {  
                System.out.println(e);  
            }  
        }  
    }  
}
```

JVM and its advantages and disadvantages?

The primary advantage of Java JVM is **code compatibility** as it eases a programmer's job to write code only once and run anywhere. Once the application is built it can be run on any device that has JVM. Apart from this it provides **security**. A program running in virtual machine is likely to suffer less from any malicious activity.

Speed and its **platform specific features** can be considered as the disadvantages of JVM. As a program needs to be translated from source code to byte code and then from byte code to executable code, the speed of execution of a program is decreased when compared to other high level languages. JVM should be free of errors as a Java program depends on JVM. Any failure of JVM leads to the **failure** of the program.

difference between an array and a linked list

An array is a collection of elements of a similar data type. A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address. Array elements store in a contiguous memory location. Linked list elements can be stored anywhere in the memory or randomly stored.

Hashset

In Java, HashSet is **commonly used if we have to access elements randomly**. It is because elements in a hash table are accessed using hash codes. The hashCode of an element is a unique identity that helps to identify the element in a hash table. HashSet cannot contain duplicate elements.

Hashmap

Allows insertion of key value pair. HashMap is non synchronized. HashMap cannot be shared between multiple threads without proper synchronization. HashMap is a fail-fast iterator. Nov 9, 2019

Different

HashMap Stores elements in form of key-value pair i.e each element has its corresponding key which is required for its retrieval during iteration. HashSet stores only objects no such key value pairs maintained. Sep 18, 2019

iterate in the context of code

programming specifically, iterative refers to a sequence of instructions or code being repeated until a specific end result is achieved.

API

the tools developers use to request data and services from outside sources.

in Java include classes, interfaces, and user Interfaces. They **enable developers to integrate various applications and websites and offer real-time information.** May 27, 2018

APIs **enable mobile experiences, connect companies on the web, and enable platform business models.** The idea of an “API economy,” in which APIs create new value for

JDK

The Java Development Kit (JDK) is a **software development environment used for developing Java applications and applets**. It includes the Java Runtime Environment

Why is Java known as the "platform-independent programming language"

Example answer: *"One of the primary purposes for Java code was to create a programming language that developers could use across multiple platforms without having to change the source code for each platform. Platform independence means the execution of your program is not dependent on the operating system being used. Earlier programming languages, such as C and C++, require developers to compile separate source code for every operating system they run it on.*

Stack vs Queue

QUEUE. The stack is a linear data structure in which insertion and deletion of elements are done by only one end. The Queue is a linear data structure in which insertion and deletion are done from two different ends i.e., rear and front ends respectively. Jan 17, 2022

Stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. Queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle.

Reverse

String Reverse

```
// reverseStr("apple") -> elppa
// reverseStr("John") -> nhoJ
// reverseStr("phone") -> enohp
// reverseStr("123456") -> "654321"

public String reverseStr(String str) {
    // create variable to store reversed version of str
    StringBuilder reverse = new StringBuilder();

    // iterate over input string from the back with charAt
    for(int i = str.length() - 1; i >= 0; i--) {
        // add chars to reversed variable
        reverse.append(String.valueOf(str.charAt(i)));
    }

    // convert to string and return reversed version
    return reverse.toString();
}

// Time Complexity: O(n)
// Using StringBuilder will be more efficient than String
// String is immutable and concatenation in the loop will create many String objects
```

Write a Java program to reverse a string without using string inbuilt function

```
public class FinalReverseWithoutUsingStringMethods {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        String str = "Automation";  
        StringBuilder str2 = new StringBuilder();  
        str2.append(str);  
        str2 = str2.reverse(); // used string builder to reverse  
        System.out.println(str2);  
    }  
}
```

String reverse

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to
        STDOUT. Your class should be named Solution. */
        Scanner scan = new Scanner(System.in);
        String A = scan.nextLine();
        System.out.println( A.equals( new StringBuilder(A).reverse()
.tostring()) ? "Yes" : "No" );
    }
}
```

Array Reverse

```
// revArr({1, 2, 3, 4, 5}) -> {5, 4, 3, 2, 1}
// revArr({1})           -> {1}
// revArr({1, 2, 3})     -> {3, 2, 1}

public void revArr(int[] arrNum) {
    // we will use two 'pointers'. One pointer will start from the beginning
    // another one from the back and we will swap their values

    // pointer that will start from the back
    int j = arrNum.length - 1;

    // our loop will go till half of our input array
    // 'i' is a pointer that will start from the beginning
    for(int i = 0; i < arrNum.length / 2; i++) {
        // swap elements using positions of i and (j - i)
        int tmp = arrNum[i];
        arrNum[i] = arrNum[j - i];
        arrNum[j - i] = tmp;
    }
}

// Time Complexity: O(n/2) but basically we can say O(n)
// Note:
// Array works with references(same objects) so if we will modify argument
// it will have effect on an original array.
```

ArrayList

```
public ArrayList<Object> reverse(ArrayList<Object> list) {  
    for(int i = 0, j = list.size() - 1; i < j; i++) {  
        list.add(i, list.remove(j));  
    }  
    return list;  
}
```

Word

```
// revWords("apple banana kiwi") -> "kiwi banana apple"
// revWords("I am John Doe")      -> "Doe John am I"
// revWords("orange")            -> "orange"

public String revWords(String str) {
    StringBuilder reversedWords = new StringBuilder();

    // split input string by " " space to get each word as String[]
    String[] words = str.split(" ");

    // loop over the array from back
    for(int i = words.length - 1; i >= 0; i--) {
        // add words to reversedWords with space
        reversedWords.append(words[i] + " ");
    }

    // trim needed to remove last space in the end
    return reversedWords.toString().trim();
}

// Time Complexity: O(n)
// Using StringBuilder will be more efficient than String
// String is immutable and concatenation in the loop will create many String objects
```

Palindrome

4. String palindrome. A palindrome is a word, phrase, number, or sequence of words that reads the same backward as forward.

String

```
// isPal("anna") -> true
// isPal("civic") -> true
// isPal("apple") -> false
// isPal("level") -> true

public boolean isPal(String str) {
    // we will use two 'pointers'. One pointer will start looking from beginning
    // another from the back. If values of pointers are not equal, we can return false

    int j = str.length() - 1; // pointer for the back

    // loop will go till half because we have two pointers
    for(int i = 0; i < str.length() / 2; i++) {
        // if pointers values are not equal return false
        if(str.charAt(i) != str.charAt(j - i)){
            return false;
        }
    }

    // if program reach here, it means all values were equal so it's palindrome
    return true;
}

// Time Complexity: O(n/2) but we can say just O(n)
```

Number

words that reads the same backward as forward. The straight forward solution would be to convert number to string and use the above approach. Some interviewers will not allow it. So let's take a look at what we can do here.

```
// isPalNum(545)    -> true
// isPalNum(1001)   -> true
// isPalNum(13)     -> false
// isPalNum(33)     -> true

public boolean isPalNum(int num) {
    // there are two most important things to remember
    // to get most right number, we can do 'num % 10'
    // to remove most right number, we can do 'num / 10'
    // both will work for any numbers

    int copyOfOriginal = num;
    int reversedNumber = 0;
    int remainder;

    while(num > 0) {
        // get most right number
        remainder = num % 10;

        // multiply by 10 to concat, not to add(plus)
        reversedNumber = (reversedNumber * 10) + remainder;

        // remove most right number from num.
        num = num / 10;
    }
}
```

```
// if reversed version and original are equal so it's palindrome
return reversedNumber == copyOfOriginal;
}

// Time Complexity: O(n)
// Note:
// 'remainder = num % 10;'
// here when num will become less than 10 expression will return that number
// so that's how program copying last(first) number

// 'num = num / 10;'
// here when num will become less than 10 expression will make it 0
// and program will exit from the loop
```

Array

An array is a **container object that holds a fixed number of values of a single type**. The length of an array is established when the array is created. After creation, its length is fixed.

Max/min number

```
// max({4, 781, 8, 99, 103})    -> 781
// max({1, 2, 3, 4, 5})          -> 5
// max({3, 4})                  -> 4
// max({100})                   -> 100

public int max(int[] arrNum) {
    // assume first element of array is biggest number
    int max = arrNum[0];

    // loop over the array and test our above assumption
    for(int i = 0; i < arrNum.length; i++) {
        // if max was not the biggest number, update it
        if(max < arrNum[i]) {
            max = arrNum[i];
        }
    }

    // after the loop max variable will hold the biggest number
    return max;
}

// Time Complexity: O(n)
```

```
// min({4, 781, 8, 99, 103}) -> 4
// min({1, 2, 3, 4, 5})      -> 1
// min({3, 4})              -> 3
// min({100})               -> 100

public int min(int[] arrNum) {
    // assume first element of array is the smallest number
    int min = arrNum[0];

    // loop over the array and test assumption
    for(int i = 0; i < arrNum.length; i++) {
        // if min was not smallest, update it
        if(min > arrNum[i]) {
            min = arrNum[i];
        }
    }

    return min;
}

// Time Complexity: O(n)
```

Find the second min/max number

```
import java.util.Arrays;

public class MaxMin {
    public static void main(String[] args) {
        System.out.println(secMax(new int[] {4, 781, 8, 99, 103})); // 103
        System.out.println(secMax(new int[] {1, 2, 3, 4, 5})); // 4
        System.out.println(secMax(new int[] {3, 4})); // 3

        System.out.println("-----");
        System.out.println(secMin(new int[] {4, 781, 8, 99, 103})); // 8
        System.out.println(secMin(new int[] {1, 2, 3, 4, 5})); // 2
        System.out.println(secMin(new int[] {3, 4})); // 4

        System.out.println("-----");
        System.out.println(nMax(new int[] {4, 3, 2, 8, 9, 5}, 1)); // 9
        System.out.println(nMax(new int[] {4, 3, 2, 8, 9, 5}, 2)); // 8
        System.out.println(nMax(new int[] {4, 3, 2, 8, 9, 5}, 3)); // 5
        System.out.println(nMax(new int[] {4, 3, 2, 8, 9, 5}, 100)); // 0
    }
}
```

```
/**  
 * secMax({4, 781, 8, 99, 103}) -> 103  
 * secMax({1, 2, 3, 4, 5}) -> 4  
 * secMax({3, 4}) -> 3  
 */  
  
public static int secMax(int[] arr) {  
    // 1. Find index of the biggest element  
    int maxIndex = 0;  
    int max = arr[0];  
    for (int i = 0; i < arr.length; i++) {  
        if (max < arr[i]) {  
            max = arr[i];  
            maxIndex = i;  
        }  
    }  
  
    // assume that second max is the smallest possible value for int  
    // it is done to make we do not get real max by different assumption  
    int secondMax = Integer.MIN_VALUE;  
    for (int i = 0; i < arr.length; i++) {  
        // i not equals to maxIndex will ensure not to get the biggest element.  
        if (i != maxIndex && secondMax < arr[i]) {  
            secondMax = arr[i];  
        }  
    }  
}
```

```
// return second biggest
return secondMax;
}

/**
 * secMin({4, 781, 8, 99, 103}) -> 8
 * secMin({1, 3, 2, 4, 5})      -> 2
 * secMin({3, 4})              -> 4
 *
 * if interviewer allows sorting it will be the easiest solution for secondMax and secondMin
 * if sorting not allowed then use above approach for second min as well.
 */
public static int secMin(int[] numArr) {
    // sort an array
    Arrays.sort(numArr);

    // return second element. Array is sorted from smallest to biggest
    return numArr[1];
}
```

```
/**  
 * Return n biggest value from array.  
 *  
 * nMax([4, 3, 2, 8, 9, 5], 1); -> 9  
 * nMax([4, 3, 2, 8, 9, 5], 2); -> 8  
 * nMax([4, 3, 2, 8, 9, 5], 3); -> 5  
 * nMax([4, 3, 2, 8, 9, 5], 100); -> 0  
 */  
  
public static int nMax(int[] arr, int n) {  
    // handle negative case  
    if (n > arr.length || n < 1) {  
        return 0;  
    }  
    // sort array  
    Arrays.sort(arr);  
  
    // return n the biggest n value by using array length and n  
    return arr[arr.length - n];  
}  
}
```

max/min

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner in=new Scanner(System.in);
        long[] m=new long[5];
        for(int i=0;i<5;i++) {
            m[i]=in.nextLong();
        }
        Arrays.sort(m);
        long x=0;
        long y=0;
        for(int i=0;i<4;i++) {
            x+=m[i];
        }
        for(int i=1;i<5;i++) {
            y+=m[i];
        }
        System.out.println(x+" "+y);
    }
}
```

Sort array without built in sort methods

```
import java.util.Arrays;

public class SortArray {
    public static void main(String[] args) {
        int[] arrNum = {6, 5, 2, 1, 9, 10, 0};
        System.out.println(Arrays.toString(arrNum)); // [6, 5, 2, 1, 9, 10, 0]
        sSort(arrNum);
        System.out.println(Arrays.toString(arrNum)); // [0, 1, 2, 5, 6, 9, 10]

        arrNum = new int [] {6, 5, 2, 1, 9, 10, 0};
        System.out.println(Arrays.toString(arrNum)); // [6, 5, 2, 1, 9, 10, 0]
        bSort(arrNum);
        System.out.println(Arrays.toString(arrNum)); // [0, 1, 2, 5, 6, 9, 10]
    }
}
```

```
/**  
 * Selection Sort  
 * The main idea to keep finding the smallest element and put it the beginning of array.  
 */  
  
public static void sSort(int[] arr) {  
    // loop over each element of array  
    for (int i = 0; i < arr.length; i++) {  
        int mIndex = i;  
        int min = arr[i];  
  
        // find smallest index  
        for (int j = i + 1; j < arr.length; j++) {  
            if (min > arr[j]) {  
                min = arr[j];  
                mIndex = j;  
            }  
        }  
  
        // swap the values of i and smallest element.  
        int tmp = arr[i];  
        arr[i] = arr[mIndex];  
        arr[mIndex] = tmp;  
    }  
}  
  
/**  
 * Bubble Sort  
 * In bubble sort we push the biggest elements to the end of array by switching pairs of elements  
 * if they are not in correct order.  
 */
```

```
public static void bSort(int[] arr) {  
    boolean isNotSorted = true;  
    // we need this because every iteration of outer loop we don't have to check last element again  
    int len = arr.length;  
  
    // loop over the array  
    while (isNotSorted) {  
        len--;  
  
        // assume is all sorted  
        isNotSorted = false;  
  
        // loop over the array  
        for (int i = 0; i < len; i++) {  
            // take two pairs and check they are in correct order.  
            if (arr[i] > arr[i + 1]) {  
                // if not swap them  
                int tmp = arr[i];  
                arr[i] = arr[i + 1];  
                arr[i + 1] = tmp;  
  
                // if something was swapped then we need keep outer loop ON  
                // if nothing got swapped then array sorted.  
                isNotSorted = true;  
            }  
        }  
    }  
}
```

String

String Pool and == operator to compare

```
public class Words {  
    public static void main(String[] args){  
        String str = "apple";  
        String str1 = "apple";  
        String str2 = new String("apple");  
  
        System.out.println(str == str1);  
        System.out.println(str == str2);  
        System.out.println(str.equals(str2));  
    }  
}
```

/*

Output:

true

false

true

Line 8: 'true' because str and str1 are both pointing to the same object.

String will reuse same objects in the String Pool. The == operator when used with references(objects) compares if they are pointing to the same object or not.

Line 9: 'false' because when we create String with new keyword it will not use String Pool and the references are pointing to different objects.

Line 10: 'true' because equals() method always compares actual value of strings

*/

}

Two string anagram

11. Two string anagram. An anagram is when all the letters in one string exist in another but the order of letters does not matter. Write a method that accepts two string arguments and returns true if they are anagram and false if they are not.

```
// isAnagram("listen", "silent")      -> true
// isAnagram("triangle", "integral") -> true
// isAnagram("abc", "bca")           -> true
// isAnagram("abc", "ccb")           -> false
// isAnagram("aaa", "aaab")          -> false

public boolean isAnagram(String str, String str1) {
    // convert both String to char[]
    char[] arrStr = str.toCharArray();
    char[] arrStr1 = str1.toCharArray();

    // sort both char[] arrays
    Arrays.sort(arrStr);
    Arrays.sort(arrStr1);

    // compare sorted arrays. If sorted arrays are equal, two strings are anagram
    return Arrays.equals(arrStr, arrStr1);
}

// Time Complexity: O(n log n)
```

Remove duplicates

```
// removeDup("hello") -> "helo"  
// removeDup("apple") -> "aple"  
// removeDup("aaaaaa") -> "a"  
// removeDup("abc") -> "abc"  
  
public String removeDup(String str) {  
    String strNoDup = "";  
  
    // loop over string and get each char  
    for(char ch : str.toCharArray()) {  
        // if strNoDup does not contain char then add to it  
        if(!strNoDup.contains(String.valueOf(ch))) {  
            strNoDup += ch;  
        }  
    }  
  
    return strNoDup;  
}  
  
// Time Complexity: O(n^2) because contains() method of String has O(n)
```

```
public static String removeDup2(String str) {  
    String strNoDup = "";  
  
    // convert str to char[]  
    char[] letters = str.toCharArray();  
    Set<Character> set = new LinkedHashSet<>();  
  
    // add each letter to set. It will remove duplicates - Set does not allow duplicates  
    for(char ch : letters) {  
        set.add(ch);  
    }  
  
    // put back to String from Set  
    for(Character ch : set) {  
        strNoDup += ch;  
    }  
  
    return strNoDup;  
}  
  
// Time Complexity: O(n)
```

Map

13. Count letters(Map). Write a method that accepts a string as an argument. The method counts the number of appearances of each char and return map. The key will be a letter and the value will be a number of appearances in the string. See input and output in the example.

Count letters

```
// countLetters("hello")      -> {h=1, e=1, l=2, o=1}
// countLetters("aaauuchhh")  -> {a=2, u=2, c=1, h=3}
// countLetters("aaaaaaa")    -> {a=6}
// countLetters("abc")        -> {a=1, b=1, c=1}

public Map<Character, Integer> countLetters(String str) {
    // if order is matter, we can use LinkedHashMap instead
    Map<Character, Integer> letters = new LinkedHashMap<>();

    for(int i = 0; i < str.length(); i++) {
        char ch = str.charAt(i);

        // if map already contains the key, get the value and put back with +1
        if(letters.containsKey(ch)) {
            letters.put(ch, letters.get(ch) + 1);
        }else {
            // if does not contains char as key, new letter put with value 1
            letters.put(ch, 1);
        }
    }

    return letters;
}

// Time Complexity: O(n)
```

Java Map

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void main(String[] args) {
        try {
            int no_of_entries = 0;
            int i = 0;
            String name = null;
            int number = 0;
            String query = null;
            HashMap<String, Integer> phoneBook = new HashMap<String,
Integer>();
            BufferedReader b = new BufferedReader(new InputStreamReader(
(
                System.in));
            no_of_entries = Integer.parseInt(b.readLine());
            while (i < no_of_entries) {
                name = b.readLine();
                number = Integer.parseInt(b.readLine());
                phoneBook.put(name, number);
                i++;
            }
            while (!(query = b.readLine().trim()).isEmpty()) {
                if (phoneBook.containsKey(query))
                    System.out.println(query + "=" + phoneBook.get
(query));
                else
                    System.out.println("Not found");
            }
        } catch (Exception e) {
        }
    }
}
```

Balanced

```
// isBalanced("[{}{}])" - true
// isBalanced("[({})]" - false
// isBalanced("{[]}" - false
// isBalanced("({}{})([{}]))" - true
// isBalanced("({") - false
public static boolean isBalanced(String str) {
    // 'open parentheses' - { [ (
    // 'close parentheses' - } ]
    // We will use stack to monitor last 'open parentheses'
    Stack<Character> st = new Stack<>(); // stack is LIFO (Last In, First Out) data structure

    // loop over input string
    for (char ch : str.toCharArray()) {
        // if char is 'open parentheses' push to our stack
        if (ch == '{' || ch == '(' || ch == '[') {
            st.push(ch);

        // if char is not open it might be 'close parentheses'
        } else {
            if (st.isEmpty()) {
                return false;
            }
        }
    }
}
```

```
// get latest 'open parentheses'  
char latestOpenedPar = st.pop();  
  
// checking here if char was 'close parentheses' then latest 'open parentheses'  
// should be appropriate to close one.  
if (latestOpenedPar == '{' && ch != '}') { // this for curly braces  
    return false;  
} else if (latestOpenedPar == '(' && ch != ')') { // this for parentheses  
    return false;  
} else if (latestOpenedPar == '[' && ch != ']') { // for square braces  
    return false;  
}  
  
// note, there is no else because if no condition match then we can continue check  
}  
}  
  
// make sure stack is empty, if not it's not balanced(for last input in the example)  
return st.size() == 0;  
}
```

How do you reverse a string in Java

```
public class StringPrograms {  
  
    public static void main(String[] args) {  
  
        String str = "123";  
        System.out.println(reverse(str));  
    }  
  
    public static String reverse(String in) {  
        if (in == null)  
            throw new IllegalArgumentException("Null is not valid input");  
  
        StringBuilder out = new StringBuilder();  
  
        char[] chars = in.toCharArray();  
  
        for (int i = chars.length - 1; i >= 0; i--)  
            out.append(chars[i]);  
  
        return out.toString();  
    }  
}
```

Java Strings Introduction

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String A=sc.next();
        String B=sc.next();
        System.out.println(A.length()+B.length());
        System.out.println(A.compareTo(B)>0?"Yes":"No");
        System.out.println(A.substring(0, 1).toUpperCase()+A.substring
(1, A.length())+" "+B.substring(0, 1).toUpperCase()+B.substring(1,
B.length())));
    }
}
```

Variables

Swap values of two

```
public class Swap {  
    public static void main(String[] args) {  
        int j = 15;  
        int i = 10;  
  
        // TODO swap values of i and j without creating any variables  
  
        j = j - i; // j = 15 - 10; j = 5  
        i = i + j; // i = 10 + 5; i = 15  
        j = i - j; // j = 15 - 5; j = 10  
  
        System.out.println(i); // 15  
        System.out.println(j); // 10  
    }  
}
```

FizzBuzz

14. FizzBuzz. Print numbers from 1 to 100

- if a number is divisible by 3 print *Fizz*
- if a number is divisible by 5 print *Buzz*
- if a number is divisible by both 3 and 5 print *FizzBuzz*

```
public void fizzBuzz() {  
    for(int i = 1; i <= 100; i++) {  
        // first check if number is divisible by 3 and 5  
        if(i % 3 == 0 && i % 5 ==0) {  
            System.out.println("FizzBuzz");  
        }else if(i % 5 == 0) {  
            System.out.println("Buzz");  
        }else if(i % 3 == 0) {  
            System.out.println("Fizz");  
        }else {  
            System.out.println(i);  
        }  
    }  
  
    // Time Complexity: O(1)  
    // Note:  
    // - Use % to check if number is divisible evenly by other number  
    // - check if number is divisible by 3 and 5 should go first
```

Even or Odd

```
// evenOrOdd(5)    -> Odd
// evenOrOdd(2)    -> Even
// evenOrOdd(100)   -> Even
// evenOrOdd(101)   -> Odd

public void evenOrOdd(int num) {
    if(num % 2 == 0) {
        System.out.println("Even");
    } else {
        System.out.println("Odd");
    }
}

// Time Complexity: O(1)
```

Sum of two

16. Sum of two. Write a method that accepts int[] array and an int number, find 2 elements in the array that sum is equal to the given int. Assume that an input array will have only one pair of numbers that sum equal to our given number. It will always have this pair. See input and output examples. I use the Brute Force algorithm.

```
// sum({1, 2, 3, 5}, 4)      -> {1, 3}
// sum({7, 7, 4, 3, 8}, 7)  -> {4, 3}
// sum({13, 43, 2, 71}, 84) -> {13, 71}

public int[] sum(int numArr[], int num) {
    int[] sumNumbers = new int[2];

    for(int i = 0; i < numArr.length; i++) {
        for(int j = i + 1; j < numArr.length; j++) {
            if(numArr[i] + numArr[j] == num) {
                sumNumbers[0] = numArr[i];
                sumNumbers[1] = numArr[j];
            }
        }
    }

    return sumNumbers;
}

// Time Complexity: O(n^2)
```

The Fibonacci

17. The Fibonacci. It is a series of numbers where the next number is the sum of the previous two numbers. The first two numbers of the Fibonacci is 0 followed by 1. Write a method that will accept one int number n . The method will print n number of Fibonacci numbers.

```
// fib(3) -> 0 1 1  
// fib(5) -> 0 1 1 2 3  
// fib(6) -> 0 1 1 2 3 5  
// fib(10) -> 0 1 1 2 3 5 8 13 21 34
```

```
public void fib(int n) {  
    int numOne = 0;  
    int numTwo = 1;  
  
    for(int i = 0; i < n; i++) {  
        System.out.print(numOne + " ");  
  
        int sum = numOne + numTwo;  
        numOne = numTwo;  
        numTwo = sum;  
    }  
}
```

// Time Complexity: O(n)

s

Write a Java program for the Fibonacci series

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        int num, a = 0, b=0, c =1;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of times");
        num = in.nextInt();
        System.out.println("Fibonacci Series of the number is:");
        for (int i=0; i<=num; i++) {
            a = b;
            b = c;
            c = a+b;
            // If you want to print on the same line, use print()
            System.out.println(a + " ");
        }
    }
}

public class Fibonacci {
    public static long fib (int n) {
        if (n <= 1) {
            return n;
        }
        return fib(n - 1) + fib(n -2);
    }
}
```

Scroll up and down

Write a Java program to show scroll up and scroll down

```
package Codes;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
public class ScrollDown {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.setProperty("webdriver.chrome.driver", "C:\\webdriver\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        JavascriptExecutor js = (JavascriptExecutor) driver;
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        driver.get(" https://www.google.com ");
        WebElement element = driver.findElement(By.name("q"));
        element.sendKeys("SoftwareTestingHelp");
        element.sendKeys(Keys.ENTER);
        js.executeScript("window.scrollBy(0,1000)");
    }
}
```

Iterator

```
import java.io.*;
import java.util.*;

public class Solution {
    static Iterator func(ArrayList mylist){

        Iterator it=mylist.iterator();

        while(it.hasNext()){
            Object element = it.next();
            if(element instanceof String)
                break;
        }
        return it;
    }
}
```

```
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN. Print output to
    STDOUT. Your class should be named Solution. */
    ArrayList mylist = new ArrayList();
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt();
    int m=sc.nextInt();

    for(int i=0;i<n;i++){
        mylist.add(sc.nextInt());
    }

    mylist.add("###");

    for(int i=0;i<m;i++){
        mylist.add(sc.next());
    }

    Iterator it=func(mylist);
```

```
while(it.hasNext()){
    Object element = it.next();
    System.out.println((String)element);
}
```

Instanceof

```
import java.util.*;
```



```
class Student{}
```

```
class Rockstar{}
```

```
class Hacker{}
```

```
public class Instance0FTutorial
{
    static String count(ArrayList mylist)
    {
        int a=0,b=0,c=0;
        for(int i=0;i<mylist.size();i++)
        {
            Object element=mylist.get(i);
            if(element instanceof Student)
                a++;
            if(element instanceof Rockstar)
                b++;
            if(element instanceof Hacker)
                c++;
        }
        String ret= Integer.toString(a)+" "+ Integer.toString(b)+" "+
        Integer.toString(c);
        return ret;
    }
}
```

```
public static void main(String []argh)
{
    ArrayList mylist=new ArrayList();
    Scanner sc=new Scanner(System.in);
    int t=sc.nextInt();
    for(int i=0;i<t;i++)
    {
        String s=sc.next();
        if(s.equals("Student"))mylist.add(new Student());
        if(s.equals("Rockstar"))mylist.add(new Rockstar());
        if(s.equals("Hacker"))mylist.add(new Hacker());
    }
    System.out.println(count(mylist));
}
```

L1

Loop

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int input = sc.nextInt();

        for(int i = 1; i <= 10; i++){
            System.out.println(input + " x " + i + " = " + input*i);
        }
    }
}
```

Java Int to String

```
import java.util.*;
import java.security.*;
public class Solution {
    public static void main(String[] args) {
```

```
        DoNotTerminate.forbidExit();
```

```
    try {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        in.close();
        //String s=???; Complete this line below

        //Write your code here
        String s = Integer.toString(n);

        if (n == Integer.parseInt(s)) {
            System.out.println("Good job");
        } else {
            System.out.println("Wrong answer.");
        }
    } catch (DoNotTerminate.ExitTrappedException e) {
        System.out.println("Unsuccessful Termination!!!");
    }
}
```

```
//The following class will prevent you from terminating the code using
//exit(0)!

class DoNotTerminate {

    public static class ExitTrappedException extends SecurityException {

        private static final long serialVersionUID = 1;
    }

    public static void forbidExit() {
        final SecurityManager securityManager = new SecurityManager() {
            @Override
            public void checkPermission(Permission permission) {
                if (permission.getName().contains("exitVM")) {
                    throw new ExitTrappedException();
                }
            }
        };
        System.setSecurityManager(securityManager);
    }
}
```

Anagrams

```
import java.io.*;
import java.util.*;

public class Solution {

    static boolean isAnagram(String A, String B) {
        char[] arrA = A.toUpperCase().toCharArray();
        char[] arrB = B.toUpperCase().toCharArray();
        Arrays.sort(arrA);
        Arrays.sort(arrB);

        for(int i = 0; i < A.length(); i++)
            try {
                if(Character.toUpperCase(arrA[i]) != Character.toUpperCase(arrB[i])) return false;
            } // end try
            catch(Exception e) {
                return false;
            } // end catch
        return true;
    } // end isAnagram

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        String A=sc.next();
        String B=sc.next();
        boolean ret=isAnagram(A,B);
        if(ret)System.out.println("Anagrams");
        else System.out.println("Not Anagrams");

    } // end main
} // end class
```

Three algorithms

sequence selection and loop

What is a method

A method is a **block of code which only runs when it is called**. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as functions.

Keyword

<u>abstract</u>	A non-access modifier. Used for classes and methods: An abstract class cannot be used to create objects (to access it, it must be inherited from another class). An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from)
<u>assert</u>	For debugging
<u>boolean</u>	A data type that can only store true and false values
<u>break</u>	Breaks out of a loop or a switch block
<u>byte</u>	A data type that can store whole numbers from -128 and 127
<u>case</u>	Marks a block of code in switch statements
<u>catch</u>	Catches exceptions generated by try statements
<u>char</u>	A data type that is used to store a single character
<u>class</u>	Defines a class
<u>continue</u>	Continues to the next iteration of a loop
<u>const</u>	Defines a constant. Not in use - use <u>final</u> instead
<u>default</u>	Specifies the default block of code in a switch statement
<u>do</u>	Used together with while to create a do-while loop

<u>double</u>	A data type that can store whole numbers from 1.7e-308 to 1.7e+308
<u>else</u>	Used in conditional statements
<u>enum</u>	Declares an enumerated (unchangeable) type
<u>exports</u>	Exports a package with a module. New in Java 9
<u>extends</u>	Extends a class (indicates that a class is inherited from another class)
<u>final</u>	A non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override)
<u>finally</u>	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
<u>float</u>	A data type that can store whole numbers from 3.4e-038 to 3.4e+038
<u>for</u>	Create a for loop
<u>goto</u>	Not in use, and has no function
<u>if</u>	Makes a conditional statement
<u>implements</u>	Implements an interface
<u>import</u>	Used to import a package, class or interface
<u>instanceof</u>	Checks whether an object is an instance of a specific class or an interface
<u>int</u>	A data type that can store whole numbers from -2147483648 to 2147483647
<u>interface</u>	Used to declare a special type of class that only contains abstract methods
<u>long</u>	A data type that can store whole numbers from -9223372036854775808 to 9223372036854775808
<u>module</u>	Declares a module. New in Java 9
<u>native</u>	Specifies that a method is not implemented in the same Java source file (but in another language)
<u>new</u>	Creates new objects
<u>package</u>	Declares a package
<u>private</u>	An access modifier used for attributes, methods and constructors, making them only accessible within the declared class
<u>protected</u>	An access modifier used for attributes, methods and constructors, making them accessible in the same package and subclasses
<u>public</u>	An access modifier used for classes, attributes, methods and constructors, making them accessible by any other class
<u>requires</u>	Specifies required libraries inside a module. New in Java 9
<u>return</u>	Finished the execution of a method, and can be used to return a value from a method
<u>short</u>	A data type that can store whole numbers from -32768 to 32767

<u>static</u>	A non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class
<u>strictfp</u>	Restrict the precision and rounding of floating point calculations
<u>super</u>	Refers to superclass (parent) objects
<u>switch</u>	Selects one of many code blocks to be executed
<u>synchronized</u>	A non-access modifier, which specifies that methods can only be accessed by one thread at a time
<u>this</u>	Refers to the current object in a method or constructor
<u>throw</u>	Creates a custom error
<u>throws</u>	Indicates what exceptions may be thrown by a method
<u>transient</u>	A non-accesss modifier, which specifies that an attribute is not part of an object's persistent state
<u>try</u>	Creates a try...catch statement
<u>var</u>	Declares a variable. New in Java 10
<u>void</u>	Specifies that a method should not have a return value
<u>volatile</u>	Indicates that an attribute is not cached thread-locally, and is always read from the "main memory"
<u>while</u>	Creates a while loop

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods? To reuse code: define the code once, and use it many times.

A **Class** is like an object constructor, or a "blueprint" for creating objects.

Qa and Testing

Software Testing is the most important step of the development cycle and is considered essential knowledge for any software position, but is often overlooked when preparing for interviews.

Unit

Testing your own code after you write it

Quick Summary: A test written by developers to ensure that a "unit" of code (usually a module or a function) behaves as intended

Also known as: Component Testing

fact

Detect bugs early (don't ship your code if it's already broken!)

Isolate a unit to its own independent environment to reveal
and remove extra dependencies

Can be performed both manually/automatically with a test
framework

White/Black/Grey box unit testing

example

Using basic assertions to ensure that the program state is behaving as expected

Using a test framework such as JUnit to write code to simulate automated test inputs for a function

- Be informed of the testing frameworks available to the language you will be working in to score some extra points with the interviewer

path

Testing every if-else case in your code

Integration

Quick Summary: Testing the compatibility of units

Also known as: Component/System Integration Testing

Important

Combine units and test them together

Usually performed after unit testing

Component (unit) Integration Testing

- Focus on interactions between components

System Integration Testing

- Focus on interactions between systems of components

example

Ensuring that nested React components in a website behave correctly (component integration testing)

Interacting with an API using your program without errors
(system integration testing)

Agile Development

- An iterative software development lifecycle
- Very flexible, client-oriented process
- Performs testing very frequently after every iteration of the product
 - Continuous feedback to improve software

Agile Testing

- Tests planned on whim and test processes are less structured
- Test cases are continually changing to meet changing demands
- Has a focus on software that works

SQL

A **database** is a system used to save and process data in an efficient manner. The way it stores information is usually in a well organized structure, and we can easily manipulate this data according to our requirements.

Structured Query Language)

SQL is a global standard for relational databases. Relational databases are ones that contain co-related, associative data.

Using **SQL** is the most efficient way to process data in datastore. You don't have to write lengthy code just to perform simple operations. It just takes a **query**. The basic queries like inserting, deleting, and updating take no more than a single line of code. Many consider **SQL** to be an intuitive language, and almost every database management system (DBMS) supports it.

command

-- THE BELOW IS SAMPLE SQL

```
create table RandomKeys (RandomKey int)
create table RandomKeysAttempt (RandomKey int)
```

-- generate m random keys between 1 and n

```
declare @cnt int = 0;
while @cnt < m
begin
    insert RandomKeysAttempt select rand()*n + 1
end;
```

-- eliminate duplicates

```
insert RandomKeys select distinct RandomKey from RandomKeysAttempt
```

-- as long as we don't have enough, keep generating new keys,

-- with luck (and m much less than n), this won't be necessary

```
declare @NextAttempt = rand()*n + 1;
while count(RandomKeys) < m
begin
    if not exists (select * from RandomKeys where RandomKey = NextAttempt)
begin
    insert RandomKeys select NextAttempt
end;
end;
```

-- get our random rows

```
select *
from RandomKeys r
join table t ON r.RandomKey = t.UniqueKey
```

Create

To create a table in an existing database, the `CREATE TABLE` statement is used. You can create an entirely new table or a new table from an existing table using this statement. Let's have a look at its syntax first, and then we will implement it.

▶ RUN SAMPLE CODE

RESET

SQL

```
1 CREATE TABLE TableName(column1Name dataType,  
2                         column2Name dataType,  
3                         column3Name dataType,  
4                         columnNName dataType);
```

`varchar` stands for `variable character` fields,

```
CREATE TABLE Class(StudentID int,  
                   StudentName varchar(255),  
                   CoursesEnrolled varchar(255),  
                   Attendance int);
```

Describe

You may infer from its namesake that `DESCRIBE` will be used to "describe" a table. But-- what does the term "describe a table" really mean?

Tables often contain data of different `types`. The `DESCRIBE` statement shows a column's name, type, and whether there are `null` values or not. The syntax is as follows:

TEXT/X-SQL

```
1 DESC Class;
```

The above line of code will generate the following output:

TEXT/X-SQL

1	StudentID	<code>int(11)</code>	YES	<code>NULL</code>
2	StudentName	<code>varchar(255)</code>	YES	<code>NULL</code>
3	CoursesEnrolled	<code>varchar(255)</code>	YES	<code>NULL</code>
4	Attendance	<code>int(11)</code>	YES	<code>NULL</code>

Select

SELECT Statement

→ The `SELECT.. FROM` statement part of SQL is an example of the

```
SELECT columnName, columnName, ..., columnName  
FROM tableName
```

Extract

```
SELECT * FROM tableName;
```

Insert, Delete, Update

The `INSERT`, `UPDATE`, and `DELETE` statements are used to manipulate the values stored within a `table`. Each state

```
INSERT INTO tableName (column1Name, column2Name, columnNName...)
VALUES (value1, value2, valueN...)
```

What do we do now? Well, if you were the administrator working on a spreadsheet, you would remove `80` and write `60` instead. In other words, **you would update the record**. For this purpose, the `UPDATE` statement is used. We can use the following query:

TEXT/X-SQL

```
1 UPDATE Class SET Attendance = 60 WHERE StudentName= 'Olivia';
```

Now let's say Olivia has left the class entirely. We no longer need her attendance record. In this case, you'll have to delete her record. Let's use the `DELETE` statement to delete Olivia's row completely.

TEXT/X-SQL

```
1 DELETE FROM Class WHERE StudentName= 'Olivia';
```

Range with Between

The `BETWEEN` operator is used to perform an operation for a specified range. The correct syntax to use this operator is attached. If you look at the query, you will notice that to apply the `BETWEEN` statement, we have applied the `SELECT / FROM` and `WHERE` statements first.

This is possible using the `WHERE` clause. What if you wanted to know Wednesday's weather from `12:00 PM` to `06:00 PM`? In this case, you've successfully defined a range for your extracted data.

```
SELECT columnName From tableName WHERE statement  
BETWEEN value1 AND value2
```

Logical

```
SELECT studentname  
FROM class  
WHERE attendance > 80  
      AND coursesenrolled = 'English'
```

Both the `HAVING` and `WHERE` clauses perform **filtering** functions.
The difference lies in *how* they perform the operation.

The difference between where and having

The `WHERE` clause performs its operation on single rows and cannot be used with the aggregate functions. On the other hand, the `HAVING` clause works *on grouped data* and that's why it's used with the aggregate functions.

```
SELECT studentname  
FROM class  
WHERE attendance > 80
```

```
SELECT studentname,  
      Sum(attendance) AS TotalAttendance  
FROM class  
GROUP BY studentname  
HAVING totalattendance > 80
```

Aggregate

An **aggregate function** is a function that's applied on multiple values, and **returns a single value** after the calculation. In **SQL**, there are multiple aggregate functions. The most commonly used ones are:

- MIN()
- MAX()
- SUM()
- AVG()
- COUNT()

Min

The `MIN()` function is used to find the row with the minimum value of a column. For example, if you want to check the minimum attendance then you'd use the following query.

▶ **RUN SAMPLE CODE**

RESET

SQL 

```
1 SELECT MIN(Attendance) FROM Class;
```

Max

The `MAX()` function is used to find the maximum value of a column. For example, if you wanted to check the maximum attendance, then you'd use the following query.

▶ RUN SAMPLE CODE

RESET

SQL 

```
1 SELECT MAX(Attendance) FROM Class;
```

Sum

The `SUM()` function is used to find the sum (the total amount resulting from the addition of two or more numbers) of values of a column. For example, if you want to find the sum of the attendance, you can do the following.

▶ **RUN SAMPLE CODE**

RESET

SQL

```
1 SELECT SUM(Attendance) FROM Class;
```

Avg

The `AVG()` function is used to find the mean, or average, value of a column. For example, if you want to find the average of the attendance, we can run this query.

▶ RUN SAMPLE CODE

RESET

SQL

```
1 SELECT MAX(Attendance) FROM Class;
```

Count

The `COUNT()` function is used to find the number of rows of a column. In other words, it counts the total number of records. Note that by default, it does not ignore `null` values.

For example, if you want to check the number of student names, you can use the following query.

▶ RUN SAMPLE CODE

RESET

SQL 

```
1 SELECT COUNT(StudentName) FROM Class;
```

Reference On Join

combine the rows of multiple columns to execute your query. The **JOIN** clause is used in this case to join two or more tables based on a mutual column.

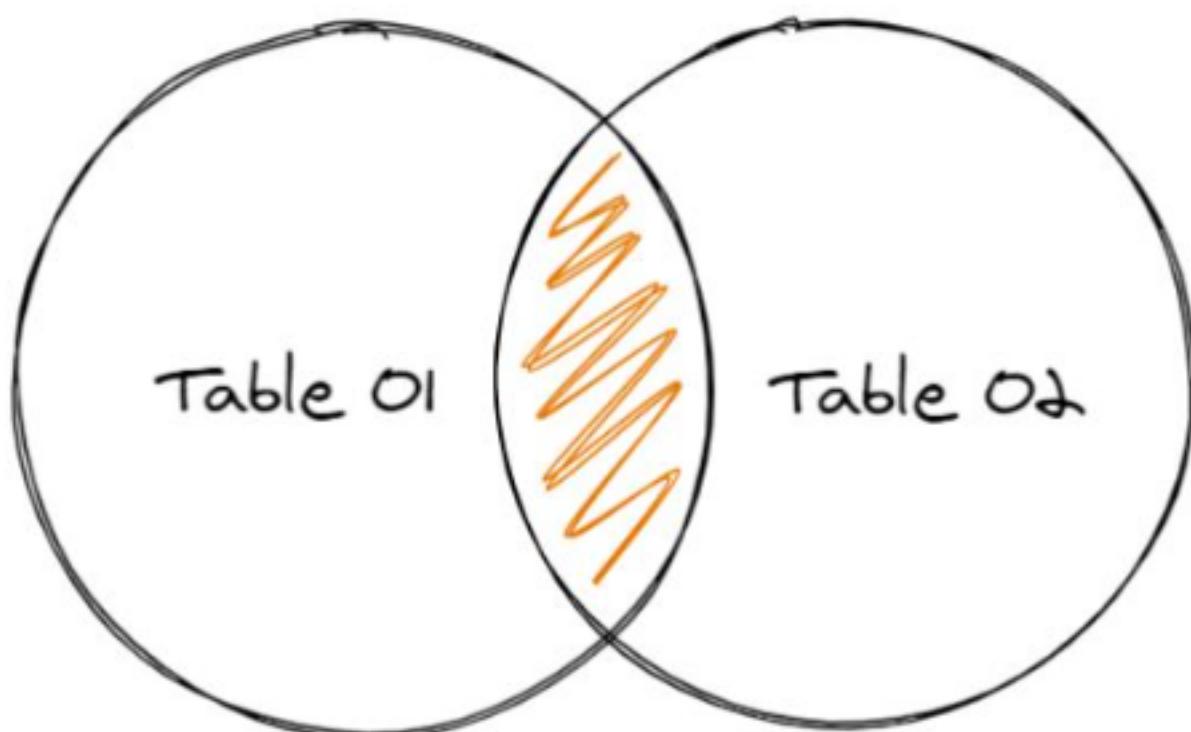
There are four types of joins:

1. INNER JOIN
2. RIGHT OUTER JOIN
3. LEFT OUTER JOIN
4. FULL OUTER JOIN

Inner

The `INNER JOIN` clause is used when we need the records which have the same value of the mutual column in both the tables.

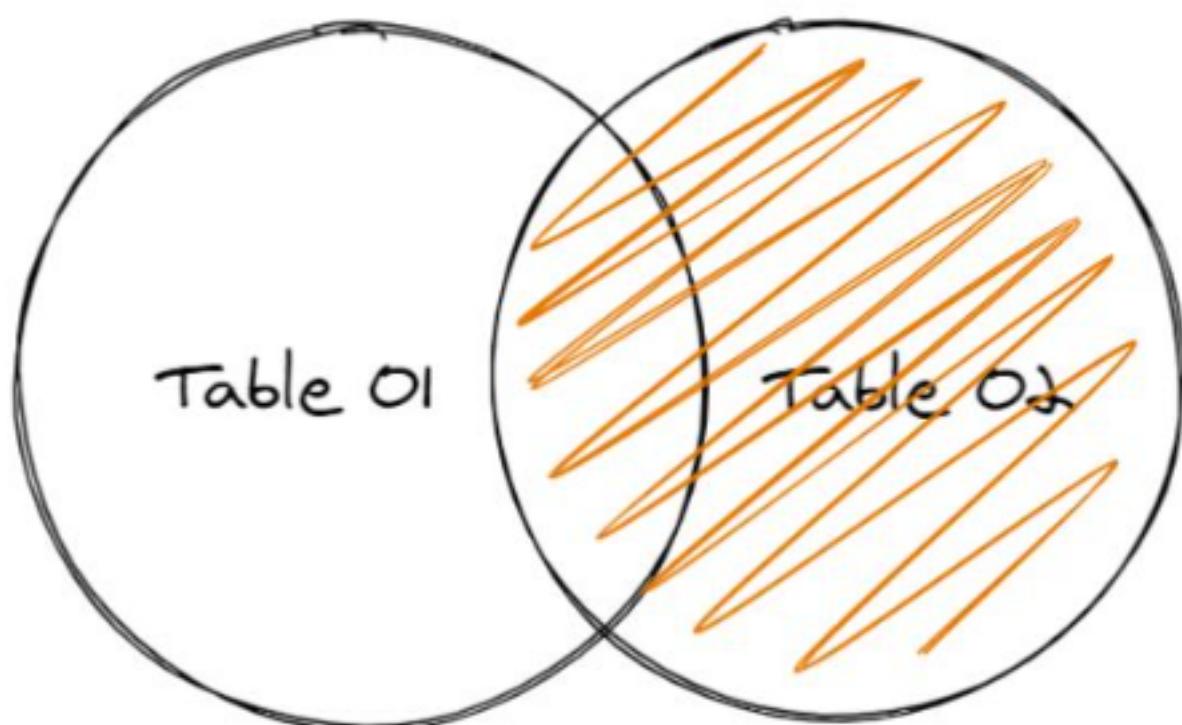
INNER JOIN



Right Outer

`RIGHT OUTER JOIN` is used when we wish to have all records from the "right" table, and just the records which have the same value of the mutual column from the left table.

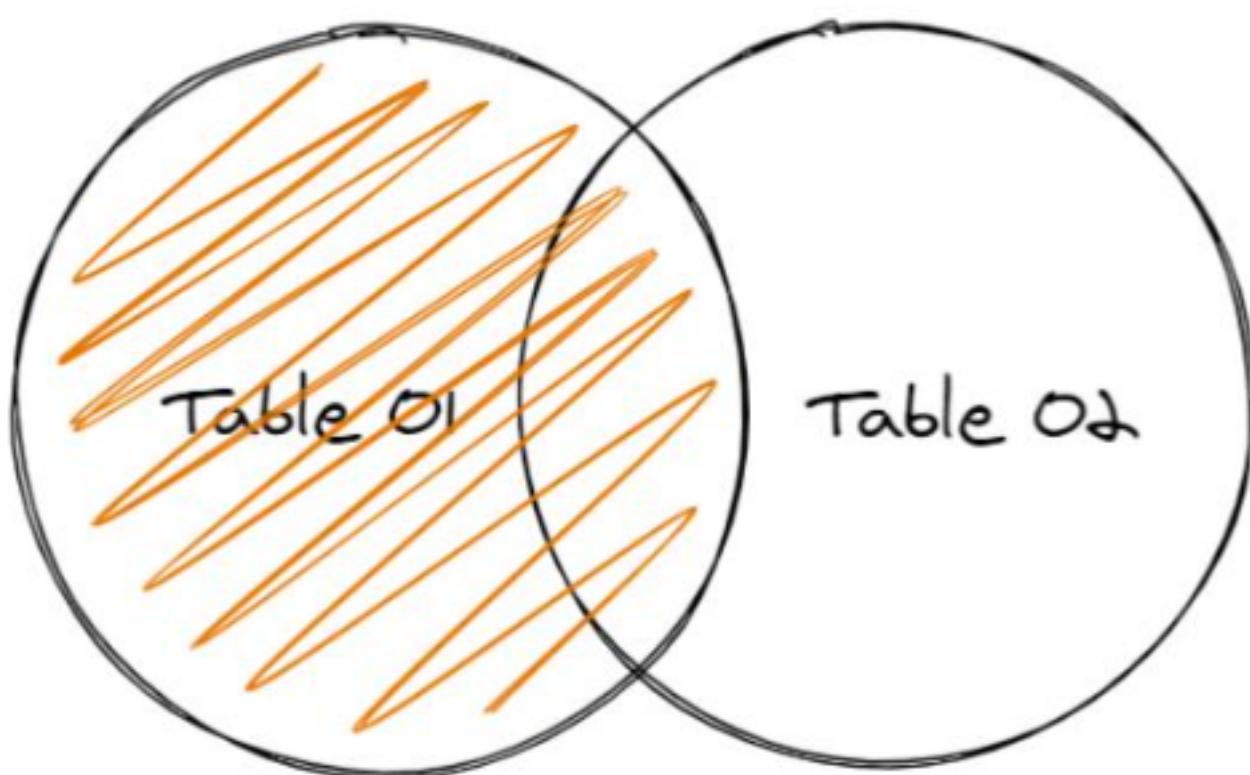
RIGHT OUTER JOIN



left Outer

LEFT OUTER JOIN is used when we wish to have all records from the left table, and just the records which have the same value of the mutual column from the right table.

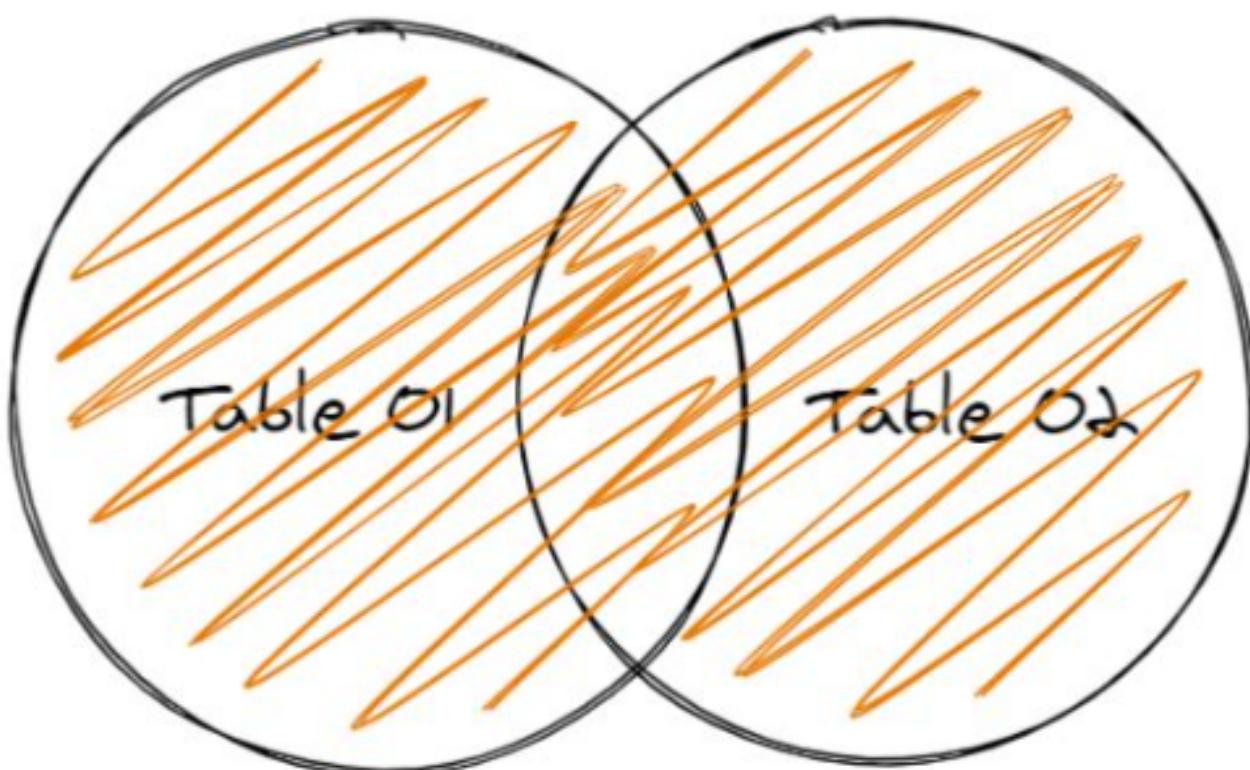
LEFT OUTER JOIN



Full Outer

FULL OUTER JOIN is used when we wish to have all records from both the tables, whether the records have the same value of the mutual column or not.

FULL OUTER JOIN



SDET

Tester: A tester is someone who performs software testing in order to find flaws. The tester additionally examines several features of the software. The tester is unaware of the software development process. A tester examines the software to see whether it contains any faults or flaws.

- a combination of an intermediate between a pure developer role and a pure tester role.
- SDETs are skilled professionals in both:
 - Quality Engineering
 - Software Development.

Parameter	SDET	Manual Tester
Testing Scope	Focuses on a wide variety of testing techniques and types. Example: Functional, Non functional, Security, Performance etc.	Generally, focus on the functionality perspective of the application under test. A manual tester behaves like a user/customer of the app under test and validates it from that perspective.
Automation	SDETs mostly focus on automating repetitive scenarios to ensure that manual testers can focus on more complex and edge scenarios and use their bandwidth and skills more efficiently.	Manual testers possess some or no skills for automation. However, it's required that manual testers should be aware of using tools that assist in manual testing Example: Using Postman for executing API endpoints, using cloud providers like sauce labs for executing tests on different platform versions, etc.

<https://apprenticenow.exeterlms.com/user/64A252FF-6EBE-4709-BACD-D328D9C1EF63>

64A252FF-6EBE-4709-BACD-D328D9C1EF6364A252FF-6EBE-4709-BACD-D328D9C1EF63

Different between SDET and manual Testing

SDET refers to a tester who is also a coder.	A tester tests software or systems after they have been developed.
SDET is well-versed in the areas of design, implementation, and testing.	A tester is unaware of the software's design and implementation.
SDET also examines the software's performance.	The tester is only responsible for testing duties.
SDET is well-versed in software requirements and other related topics.	A tester has a limited understanding of software requirements.
SDET is involved at every stage of the software development life cycle.	In the software development life cycle, testers play a less role and have fewer obligations.

ad-Hoc

Ad hoc testing is a type of unstructured or informal software testing that seeks to interrupt the testing process in order to uncover potential defects or errors as soon as feasible. Ad hoc testing is a type of testing that is done at random and is usually an unplanned activity that does not use any documentation or test design methodologies to construct test cases.

No documentation

No Test Design

No Test Case

Code Inspection

role and responsibilities

Work alongside developers as well as the business stakeholders and strive to automate the acceptance criteria. This means in simple words is – an SDET first understands the requirements from an acceptance/customer perspective and also has to understand the way the product is developed in terms of coding language, databases, etc, and then plans a strategy to automate maximum scenarios as possible.

Responsible for building robust, and high-quality test automation solutions for functional, regression, and performance testing.

Create reusable scripts/tools wherever required.

Contribute to both functional and non-functional areas of testing. Functional testing includes testing from a functionality/requirements perspective and is largely driven by acceptance criteria or user stories.

Skillset

Should have a solid understanding of testing principles, testing types, and methodologies.

Highly proficient at debugging issues – learn debug tools like – [Chrome Web Debugger](#) which are extremely helpful for debugging web applications, as well as investigating network logs for an app under test.

They should be able to write reusable code/scripts and hence they should be proficient in at least one scripting language. The easiest to learn is Python which could be applied to a wide variety of tasks, automation frameworks, etc.

Be familiar with API testing clients like [POSTMAN](#)

Should be aware of White box testing tools and techniques – like Mocking frameworks ([Mockito](#)), etc as they might be expected to contribute to writing unit tests as well when required.

They should be aware of versioning tools like [Git](#). Also, they should be familiar with the concepts of [Pull requests](#), code reviews, etc.

Kanban vs. Scrum

With transparency at the core of the Scrum framework, it's crucial that the team builds trust and an ability to share, critique and constructively challenge each other.

Kanban	Scrum
No prescribed roles	Pre-defined roles of Scrum master, Product owner and team member
Continuous Delivery	Timeboxed sprints
Work is pulled through the system (single piece flow)	Work is pulled through the system in batches (the sprint backlog)
Changes can be made at any time	No changes allowed mid-sprint
Cycle time	Velocity
More appropriate in operational environments with a high degree of variability in priority	More appropriate in situations where work can be prioritized in batches that can be left alone

Next, you will dive deeper into the specifics of how the Kanban method can help your team.

Agile Final Exam Review

- Analysis
- Design
- Implementation
- Testing
- Deployment

Analysis

- **Figure out what you need**
- **Gather requirements**
- **Consult everyone who has stake in the project**
 - Management
 - Customer
 - Subject matter experts

Design

- What do you do to meet those requirements?
- Plan out your project

Implementation

- **Do your project**
- **Requires the most people**
- **Is the most complex**
- **Is the most subject to delays**

Testing

- Did you did a good job?
- Does everything work?
- Does it meet the requirements?
- Is the customer satisfied?
- Find and fix all the bugs / issues

Deployment

- Prepare for launch
- Put the results of your project into action

Project Management

- The application of knowledge, skills, tools, and techniques to produce activities to meet the project requirements

Do

Program Manager

- Oversee multiple large projects

Project Manager

- Oversees one project

- Resources – time and cost

- Scope

- Communication

Software

- Jira – Gold standard for Scrum
- Team Foundation (TFS) – like Jira, but less widely used and integrated with Microsoft
- Trello – Scrum, to-do lists, etc.
- Liquid Planner – Waterfall
- Microsoft Project - Waterfall

Answer

Kanban teams rely on limiting Work-In-Progress to throttle the flow of cards.

A True

Correct ✓

B False

Question 2

1 point

The team can pull a card from the top of the sprint backlog at any time.

A True

B False

Correct ✓

In which state of the Software Development Lifecycle (SDLC) do the system architects evaluate architectural strategies?

A Analysis

B Design

Correct ✓

C Implementation

D Testing

Question 4

1 point

A Kanban Board is a visual representation of the Kanban project's current status.

A True

Correct ✓

B False

What is used to describe task details and documentation on a Kanban board?

A Swim lane

B Slice

C Card

Correct ✓

D Note

Which is *NOT* a typical Kanban Board swim lane?

A To Do

B Doing

C Done

D Backlog

Correct ✓

Question 7

1 point

Project Management is *the application of knowledge, skills, tools, and techniques to produce activities to meet the project requirements.*

A True

Correct ✓

B False

Which pillar of Kanban helps the team to avoid overcommitting?

A Visualize what you do today

B Limit the work-in-progress

Correct ✓

C Enhance flow

D Continuous improvement

Which of the following is *NOT* one of the Three Pillars of Scrum?

A Transparency

B Inspection

C Adaptation

D Consistency

Correct ✓

Question 10

2 points

What's the difference between a Product Backlog and a Sprint Backlog?

A The Product Backlog contains User Stories while the Sprint Backlog contains User Stories and feature requests.

B The Product Backlog contains both User Stories and features while the Sprint Backlog contains User Stories.

Correct ✓

Swim Lanes are task statuses (e.g., To-Do, Doing, Done).

A True

Correct ✓

B False

Question 12

2 points

Which is a pillar of Kanban that describes the pace of work?

A Visualize what you do today

B Limit the work-in-progress

C Enhance flow

Correct ✓

D Continuous improvement

Which is a pillar of Kanban that describes how we should improve efficiencies through the constant re-evaluation of our processes?

A Visualize what you do today

B Limit the work-in-progress

C Enhance flow

D Continuous improvement

Correct ✓

Question 14

1 point

A Project Manager who oversees multiple large projects is known as a Program Manager.

A True

Correct ✓

B False

A Project Manager is responsible for which of the following?

A Scheduling project resources

B Managing the scope of the project

Incorrect ✗

C Project communication

D All of the above

Correct ✓

Question 16

1 point

When choosing a system to track your Scrum team, it is advised to use a scrum management system with advanced functionality to meet the needs of the team.

A True

It is a common pitfall that teams find themselves adopting the most sophisticated system available, when a whiteboard or sticky notes may suffice while saving time and money.

Incorrect ✗

B False

Correct ✓

The essential aspects of tracking can be reduced to knowing the status of the current sprint and knowing what's in the product backlog.

A	True	Correct ✓
B	False	Incorrect ✗

Question 18

1 point

TFS offers many features found in JIRA with the added benefit of being tightly integrated with the Microsoft platform.

A	True	Correct ✓
B	False	

Which order is correct for the stages in the Waterfall methodology?

- A Design, Requirements, Implementation, Verification, Maintenance
- B Requirements, Verification, Design, Implementation, Maintenance
- C Requirements, Design, Verification, Implementation, Maintenance
- D Requirements, Design, Implementation, Verification, Maintenance

Correct ✓

Question 20

1 point

Technical documentation is thorough and complete when using the Waterfall method.

A	True	Correct ✓
B	False	

Development methodologies are recommendations for optimizing how to navigate the essential stages.

A True

Correct ✓

B False

Incorrect ✗

Question 22

1 point

Restricting the Work-In-Progress immediately identifies bottlenecks and skill gaps.

A True

Correct ✓

B False

The ultimate benefit of Kanban is passed on to the developer through a higher-quality product or service.

A True

The ultimate benefit of Kanban is passed on to the customer through a higher-quality product or service at a cost-effective price due to the decrease in waste.

Incorrect ✗

B False

Correct ✓

Question 24

1 point

The key tasks in your project that form the maximum time needed to complete the project is referred to as the **critical path**.

A True

The critical path is defined by the key tasks that form the *minimum* time needed to complete the project, not the *maximum*.

Incorrect ✗

B False

Correct ✓

Which tool can be used to provide a visual view of tasks and their relationships scheduled over time — the big picture?

A Scrum sprint

B Swim lane

C Gantt chart

Correct ✓

D None of the above

Product vs Sprint backlog

Product

- All known tickets
- Wishlist
- Goes beyond the sprint

Sprint

- Only what you can tackle in the sprint
- Pull from the product backlog to fill

Three pillar of scum

Transparency

Inspected

Adaptation

notes

Software Life Cycle

Requirements Analysis. Software organizations provide solutions to customer requirements by developing appropriate software that best suits their specifications. Thus, the life of software starts with origin of requirements. Very often, these requirements are vague, emergent and always subject to change.

Design and Specifications. The outcome of requirements analysis is the requirements specification. Using this, the overall design for the intended software is developed. Activities in this phase - Perform Architectural Design for the software, Design Database (If applicable), Design User Interfaces, Select or Develop Algorithms (If Applicable), Perform Detailed Design.

Coding. The development process tends to run iteratively through these phases rather than linearly; several models (spiral, waterfall etc.) have been proposed to describe this process. Activities in this phase - Create Test Data, Create Source, Generate Object Code, Create Operating Documentation, Plan Integration, Perform Integration

Testing. The process of using the developed system with the intent to find errors. Defects/flaws/bugs found at this stage will be sent back to the developer for a fix and have to be re-tested. This phase is iterative as long as the bugs are fixed to meet the requirements. Activities in this phase - Plan Verification and Validation, Execute Verification and validation Tasks, Collect and Analyze Metric Data, Plan Testing, Develop Test Requirements, Execute Tests

Installation. The so developed and tested software will finally need to be installed at the client place. Careful planning has to be done to avoid problems to the user after installation is done. Beginners Guide To Software Testing Activities in this phase - Plan Installation, Distribution of Software, Installation of Software, Accept Software in Operational Environment.

Operation and Support. Support activities are usually performed by the organization that developed the software. Both the parties usually decide on these activities before the system is developed. Activities in this phase - Operate the System, Provide Technical Assistance and Consulting, Maintain Support Request Log.

Maintenance. The process does not stop once it is completely implemented and installed at user place; this phase undertakes development of new features, enhancements etc. Activities in this phase - Reapplying Software Life Cycle.

Software Testing Life Cycle Software Testing Life Cycle consist of six (generic) phases: 1) Planning, 2) Analysis, 3) Design, 4) Construction, 5) Testing Cycles, 6) Final Testing and Implementation and 7) Post Implementation. Each phase in the life cycle is described with the respective activities.

Planning. Planning High Level Test plan, QA plan (quality goals), identify – reporting procedures, problem classification, acceptance criteria, databases for testing, measurement criteria (defect quantities/severity level and defect origin), project metrics and finally begin the schedule for project testing. A test cases (manual or automated) in a database.

Analysis. Involves activities that - develop functional validation based on Business Requirements (writing test cases basing on these details), develop test case format (time estimates and priority assignments), develop test cycles (matrices and timelines), identify test cases to be automated (if applicable), define area of stress and performance testing, plan the test cycles required for the project and regression testing, define procedures for data maintenance (backup, restore, validation), review documentation.

Design. Activities in the design phase - Revise test plan based on changes, revise test cycle matrices and timelines, verify that test plan and cases are in a database or requisite, continue to write test cases and add new ones based on changes, develop Risk Assessment Criteria, formalize details for Stress and Performance testing, finalize test cycles (number of test case per cycle based on time estimates per test case and priority), finalize the Test Plan, (estimate resources to support development in unit testing).

Construction (Unit Testing Phase). Complete all plans, complete Test Cycle matrices and timelines, complete all test cases (manual), begin Stress and Performance testing, test the automated testing system and fix bugs, (support development in unit testing), run QA acceptance test suite to certify software is ready to turn over to QA. Test Cycle(s) / Bug Fixes (Re-Testing/System Testing Phase). Run the test cases (front and back end), bug reporting, verification, and revise/add test cases as required.

Final Testing and Implementation (Code Freeze Phase). Execution of all front end test cases - manual and automated, execution of all back end test cases - manual and automated, execute all Stress and Performance tests, provide on-going defect tracking metrics, provide on-going complexity and design metrics, update estimates for test cases and test plans, document test cycles, regression testing, and update accordingly.

Post Implementation. Post implementation evaluation meeting can be conducted to review entire project. Activities in this phase - Prepare final Defect Report and associated metrics, identify strategies to prevent similar problems in future project, automation team - 1) Review test cases to evaluate other cases to be automated for regression testing, 2) Clean up automated test cases and variables, and 3) Review

Testing Levels and Types

There are basically three levels of testing i.e. Unit Testing, Integration Testing and System Testing. Various types of testing come under these levels.

Unit Testing

To verify a single program or a section of a single program

Integration Testing

To verify interaction between system components

Prerequisite: unit testing completed on all components that compose a system.

System Testing

To verify and validate behaviors of the entire system against the original system objectives.

Software testing is a process that identifies the correctness, completeness, and quality of software.

Following is a list of various types of software testing and their definitions in a random order:

1. Formal Testing: Performed by test engineers
2. Informal Testing: Performed by the developers
3. Manual Testing: That part of software testing that requires human input, analysis, or evaluation.
4. Automated Testing: Software testing that utilizes a variety of tools to automate the testing process. Automated testing still requires a skilled quality assurance professional with knowledge of the automation tools and the software being tested to set up the test cases.
5. Black box Testing: Testing software without any knowledge of the back-end of the system, structure or language of the module being tested. Black box test cases are written from a definitive source document, such as a specification or requirements document.
6. White box Testing: Testing in which the software tester has knowledge of the back-end, structure and language of the software, or at least its purpose.
7. Unit Testing: Unit testing is the process of testing a particular compiled program, i.e., a window, a report, an interface, etc. independently as a stand-alone component/program. The types and degrees of unit tests can vary among modified and newly created programs. Unit testing is mostly performed by the programmers who are also responsible for the creation of the necessary unit test data.
8. Incremental Testing: Incremental testing is partial testing of an incomplete product. The goal of incremental testing is to provide an early feedback to software developers.
9. System Testing: System testing is a form of black box testing. The purpose of system testing is to validate an application's accuracy and completeness in performing the functions as designed.
10. Integration Testing: Testing two or more modules or functions together with the intent of finding interface defects between the modules/functions.
11. System Integration Testing: Testing of software components that have been distributed across multiple platforms (e.g., client, web server, application server, and database server) to produce failures caused by system integration defects (i.e. defects involving distribution and back-
12. Functional Testing: Verifying that a module functions as stated in the specification and establishing confidence that a program does what it is supposed to do.
13. End-to-end Testing: Similar to system testing - testing a complete application in a situation that mimics real world use, such as interacting with a database, using network communication, or interacting with other hardware, application, or system.
14. Sanity Testing: Sanity testing is performed whenever cursory testing is sufficient to prove the application is functioning according to specifications. This level of testing is a subset of regression testing. It normally includes testing basic GUI functionality to demonstrate connectivity to the database, application servers, printers, etc.

15. Regression Testing: Testing with the intent of determining if bug fixes have been successful and have not created any new problems.
16. Acceptance Testing: Testing the system with the intent of confirming readiness of the product and customer acceptance. Also known as User Acceptance Testing.
17. Adhoc Testing: Testing without a formal test plan or outside of a test plan. With some projects this type of testing is carried out as an addition to formal testing. Sometimes, if testing occurs very late in the development cycle, this will be the only kind of testing that can be performed – usually done by skilled testers. Sometimes ad hoc testing is referred to as exploratory testing.
18. Configuration Testing: Testing to determine how well the product works with a broad range of hardware/peripheral equipment configurations as well as on different operating systems and software.
19. Load Testing: Testing with the intent of determining how well the product handles competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.
20. Stress Testing: Testing done to evaluate the behavior when the system is pushed beyond the breaking point. The goal is to expose the weak links and to determine if the system manages to recover gracefully.
21. Performance Testing: Testing with the intent of determining how efficiently a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.
22. Usability Testing: Usability testing is testing for 'user-friendliness'. A way to evaluate and measure how users interact with a software product or site. Tasks are given to users and observations are made.
23. Installation Testing: Testing with the intent of determining if the product is compatible with a variety of platforms and how easily it installs.
24. Recovery/Error Testing: Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.
25. Security Testing: Testing of database and network software in order to keep company data and resources secure from mistaken/accidental users, hackers, and other malevolent attackers.
26. Penetration Testing: Penetration testing is testing how well the system is protected against unauthorized internal or external access, or willful damage. This type of testing usually requires sophisticated testing techniques.
27. Compatibility Testing: Testing used to determine whether other system software components such as browsers, utilities, and competing software will conflict with the software being tested.
28. Exploratory Testing: Any testing in which the tester dynamically changes what they're doing for test execution, based on information they learn as they're executing their tests.
29. Comparison Testing: Testing that compares software weaknesses and strengths to those of competitors' products.
30. Alpha Testing: Testing after code is mostly complete or contains most of the functionality and prior to reaching customers. Sometimes a selected group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.
31. Beta Testing: Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large.
32. Gamma Testing: Gamma testing is testing of software that has all the required features, but it did not go through all the in-house quality checks.
33. Mutation Testing: A method to determine the test thoroughness by measuring the extent to which the test cases can discriminate the program from slight variants of the program.
34. Independent Verification and Validation (IV&V): The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The individual or group doing this work is not part of the group or organization that developed the software.

35. Pilot Testing: Testing that involves the users just before actual release to ensure that users become familiar with the release contents and ultimately accept it. Typically involves many users, is conducted over a short period of time and is tightly controlled. (See beta testing)
36. Parallel/Audit Testing: Testing where the user reconciles the output of the new system to the output of the current system to verify the new system performs the operations correctly.
37. Glass Box/Open Box Testing: Glass box testing is the same as white box testing. It is a testing approach that examines the application's program structure, and derives test cases from the application's program logic.
38. Closed Box Testing: Closed box testing is same as black box testing. A type of testing that considers only the functionality of the application.
39. Bottom-up Testing: Bottom-up testing is a technique for integration testing. A test engineer creates and uses test drivers for components that have not yet been developed, because, with bottom-up testing, low-level components are tested first. The objective of bottom-up testing is to call low-level components first, for testing purposes.
40. Smoke Testing: A random test conducted before the delivery and after complete testing.

Testing Terms

41. Bug: A software bug may be defined as a coding error that causes an unexpected defect, fault or flaw. In other words, if a program does not perform as intended, it is most likely a bug.
42. Error: A mismatch between the program and its specification is an error in the program.
43. Defect: Defect is the variance from a desired product attribute (it can be a wrong, missing or extra data). It can be of two types – Defect from the product or a variance from customer/user expectations. It is a flaw in the software system and has no impact until it affects the user/customer and operational system. 90% of all the defects can be caused by process problems.
44. Failure: A defect that causes an error in operation or negatively impacts a user/ customer.
45. Quality Assurance: Is oriented towards preventing defects. Quality Assurance ensures all parties concerned with the project adhere to the process and procedures, standards and templates and test readiness reviews.
46. Quality Control: quality control or quality engineering is a set of measures taken to ensure that defective products or services are not produced, and that the design meets performance requirements.
47. Verification: Verification ensures the product is designed to deliver all functionality to the customer; it typically involves reviews and meetings to evaluate documents, plans, code, requirements and specifications; this can be done with checklists, issues lists, walkthroughs and inspection meetings.
48. Validation: Validation ensures that functionality, as defined in requirements, is the intended behavior of the product; validation typically involves actual testing and takes place after Verifications are completed.

What is a Test Strategy? What are its Components?

Test Policy - A document characterizing the organization's philosophy towards software testing.

Test Strategy - A high-level document defining the test phases to be performed and the testing within those phases for a program. It defines the process to be followed in each project. This sets the standards for the processes, documents, activities etc. that should be followed for each project.

For example, if a product is given for testing, you should decide if it is better to use black-box testing or white-box testing and if you decide to use both, when will you apply each and to which part of the software? All these details need to be specified in the Test Strategy.

Project Test Plan - a document defining the test phases to be performed and the testing within those phases for a particular project.

A Test Strategy should cover more than one project and should address the following issues: An approach to testing high risk areas first, Planning for testing, How to improve the process based on previous testing, Environments/data used, Test management - Configuration management, Problem management, What Metrics are followed, Will the tests be automated and if so which tools will be used, What are the Testing Stages and Testing Methods, Post Testing Review process, Templates.

Test planning needs to start as soon as the project requirements are known. The first document that needs to be produced then is the Test Strategy/Testing Approach that sets the high level approach for testing and covers all the other elements mentioned above.

A test case is a detailed procedure that fully tests a feature or an aspect of a feature. While the test plan describes what to test, a test case describes how to perform a particular test. You need to develop test cases for each test listed in the test plan.

Automating testing is no different from a programmer using a coding language to write programs to automate any manual process. One of the problems with testing large systems is that it can go beyond the scope of small test teams. Because only a small number of testers are available the coverage and depth of testing provided are inadequate for the task at hand.

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

- Annotations.
- Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
- Test that your code is multithread safe.
- Flexible test configuration.
- Support for data-driven testing (with @DataProvider).
- Support for parameters.
- Powerful execution model (no more TestSuite).
- Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
- Embeds BeanShell for further flexibility.
- Default JDK functions for runtime and logging (no dependencies).
- Dependent methods for application server testing.

Selenium - Webdriver, WebDriver is a tool for automating testing web applications.

Gherkin is a business readable language which helps you to describe business behavior without going into details of implementation. It is a domain specific language for defining tests in Cucumber format for specifications. It uses plain language to describe use cases and allows users to remove logic details from behavior tests.

Cucumber is a Behavior Driven Development tool used to develop test cases for the behavior of software's functionality. It plays a supporting role in automated testing.

In other words "Cucumber is a software tool used by the testers to develop test cases for the testing of behavior of the software."

Cucumber tool plays a vital role in the development of acceptance test cases for automation testing. It is mainly used to write acceptance tests for web applications as per the behavior of their functionalities.

Maven: The java build tool provided by Apache to help in the build, documentation and dependency process of projects with any level of complexity written in Java and C# that uses Project Object Model (POM) and that follows the convention of source code, compiling code.

Gradle is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging to testing, deployment, and publishing. Supported languages include Java (as well as Kotlin, Groovy, Scala), C/C++, and JavaScript.[2] It also collects statistical data about the usage of software libraries around the globe.

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.

Testing within IntelliJ

Annotations

@Test

This annotation denotes that a method is a test method. Note this annotation does not take any attributes

@ParameterizedTest

Parameterized tests make it possible to run a test multiple times with different arguments. They are declared just like regular @Test methods but use the @ParameterizedTest annotation instead.

@RepeatedTest

JUnit 5 has the ability to repeat a test a specified number of times simply by annotating a method with @RepeatedTest and specifying the total number of repetitions desired.

The `@Disabled` annotation is used to disable or skip tests at class or method level.

`@AfterAll`

The `@AfterAll` annotation is used to execute the annotated method, only after all tests have been executed

Intro to testing

JUnit is the standard for testing in Java-based applications and is supported by almost all IDEs, including Eclipse with which you are already familiar. A JUnit test is a method contained in a class only and it is only used for testing. This class is called a Test class. We define that a certain method is a test method and annotate it with the `@Test` annotation. When using the `@Test` annotation, we execute the code. The assert method checks the results according to the expected or actual result. We call these methods "asserts" or, sometimes, "assert statements." Assert statements typically start with `assert`. They allow you to specify the error message, the expected result, and the actual result.

An **assertion method** compares the actual value returned by a test to the expected value. It throws an `AssertionException` if the comparison fails. When our developer uses this method to Unit test, she should provide meaningful messages. This will assist her and, perhaps her team, to address the issue. Below is an example of a JUnit test:

This test assumes that `MyClass` exists

```
import static org.junit.jupiter.api.Assertions.assertEquals;  
  
import org.junit.jupiter.api.Test;  
  
public class MyTests {  
  
    @Test  
    public void multiplicationOfZeroIntegersShouldReturnZero() {  
        MyClass tester = new MyClass(); // MyClass is tested  
  
        // assert statements  
        assertEquals(0, tester.multiply(10, 0), "10 x 0 must be 0");  
        assertEquals(0, tester.multiply(0, 10), "0 x 10 must be 0");  
        assertEquals(0, tester.multiply(0, 0), "0 x 0 must be 0");  
    }  
}
```

Example

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

public class SampleTest {

    private java.util.List emptyList;

    /**
     * Sets up the test fixture.
     * (Called before every test case method.)
     */
    @Before
    public void setUp() {
        emptyList = new java.util.ArrayList();
    }

    /**
     * Tears down the test fixture.
     * (Called after every test case method.)
     */
    @After
    public void tearDown() {
        emptyList = null;
    }

    @Test
    public void testSomeBehavior() {
        assertEquals("Empty list should have 0 elements", 0, emptyList.size());
    }

    @Test(expected=IndexOutOfBoundsException.class)
    public void testForException() {
        Object o = emptyList.get(0);
    }
}
```

Parameters

JUnit allows parameters in a test class. This class can contain one test method and this method is executed with various parameters provided. We mark a parameterized test with the `@RunWith(Parameterized.class)` annotation. In this case, the test class must contain a static method that is annotated with a `@Parameters`. The method then generates and returns a collection. Each item included in the collection is used as a parameter for the test method. Such a test class contains a static method annotated with the `@Parameters` annotation. As you work with the `@Paramethers` annotation, you will see that you can use the `@Parameter` annotation on public fields to get test values inserted in your test.

example

```
package JUnittesting;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import java.util.Arrays;
import java.util.Collection;
import static org.junit.Assert.assertEquals;
import static org.junit.runners.Parameterized.*;
@RunWith(Parameterized.class)
public class ParameterizedTestFields {
    // fields used together with @Parameter must be public
    @Parameter(0)
    public int m1;
    @Parameter(1)
    public int m2;
    @Parameter(2)
    public int result;
    // creates the test data
    @Parameters
    public static Collection<Object[]> data() {
        Object[][] data = new Object[][][] { { 1, 2, 2 }, { 5, 3, 15 },
        { 121, 4, 484 } };
        return Arrays.asList(data);
    }
}
```

```

@在这
public void testMultiplyException() {
    MyClass tester = new MyClass();
    assertEquals("Result", result, tester.multiply(m1, m2));
}
// class to be tested
class MyClass {
    public int multiply(int i, int j) {
        return i *j;
    }
}
}

```

additional info

<https://github.com/Pragmatists/JUnitParams>

JUnit allows you to add rules to the behavior in a test class. We use TestRule with the @Rule annotation. Testers/Developers can write custom rules by invoking the TestRule interface. The TestRule interface defines the apply(Statement, Description) method which must return an instance of Statement.

To see the result of a JUnit test, Eclipse uses the JUnit view, which shows school. `setStartedDate(date);`

```

    school.setModifiedTime(date);
}

@Test
public final void testAddSchool() throws Exception {
    School newSchool = schoolService.addSchool(school);
    assertNotNull("School Type object should not null ", newSchool);
}

@Test
public final void testUpdateSchool() {
    School newSchool = schoolService.addSchool(school);
    assertNotNull(newSchool);
    newSchool.setContactNo("0145785545");
    schoolService.updateSchool(newSchool);
}

```

```
@Test  
public final void testFindSchool() throws AkuraAppException {  
  
    School newSchool = schoolService.addSchool(school);  
    assertNotNull(newSchool);  
    School findSchool = schoolService.findSchool(  
        newSchool.getSchoolId());  
    assertNotNull("School Type object should not null ", findSchool);  
}  
  
  
@Test  
public final void testGetSchoolList() throws AkuraAppException {  
    School newSchool = schoolService.addSchool(school);  
    assertNotNull(newSchool);  
    List<School> schoolList = schoolService.getSchoolList();  
}  
  
  
@After  
public final void teardown() throws SchoolNotFoundException {  
    schoolDao.delete(school);  
}  
}
```

Testing With TestNG

official link

<https://testng.org/doc/documentation-main.html>

lastest version

<http://www.testng.org/>

help doc

<https://testng.org/doc/index.html>

Keynote

Keyword	Description
Functional Testing	A type of application testing used to validate the system against the business requirements.
Automation Testing	A type of quality assurance testing involving running tests automatically, using the results to improve software quality.
CSS	A style sheet language used for describing the presentation of a document. Cascading Style Sheets are written in a markup language generally HTML.
HTML	Hypertext Markup Language used to display documents in a browser.
JUnit	A testing framework for the Java programming language.
TestNG	An automation testing framework for the Java programming language as an improvement to JUnit. TestNG or Test (Next Generation)
Test Cases	Provides test data, along with expected results developed for a particular test scenario. Test cases verify compliance against a specific requirement ie. maximum value of a field, minimum value of a field
Parallel execution of test cases	Consists of running multiple test cases

Testing Fundamental

Software Development Lifecycle

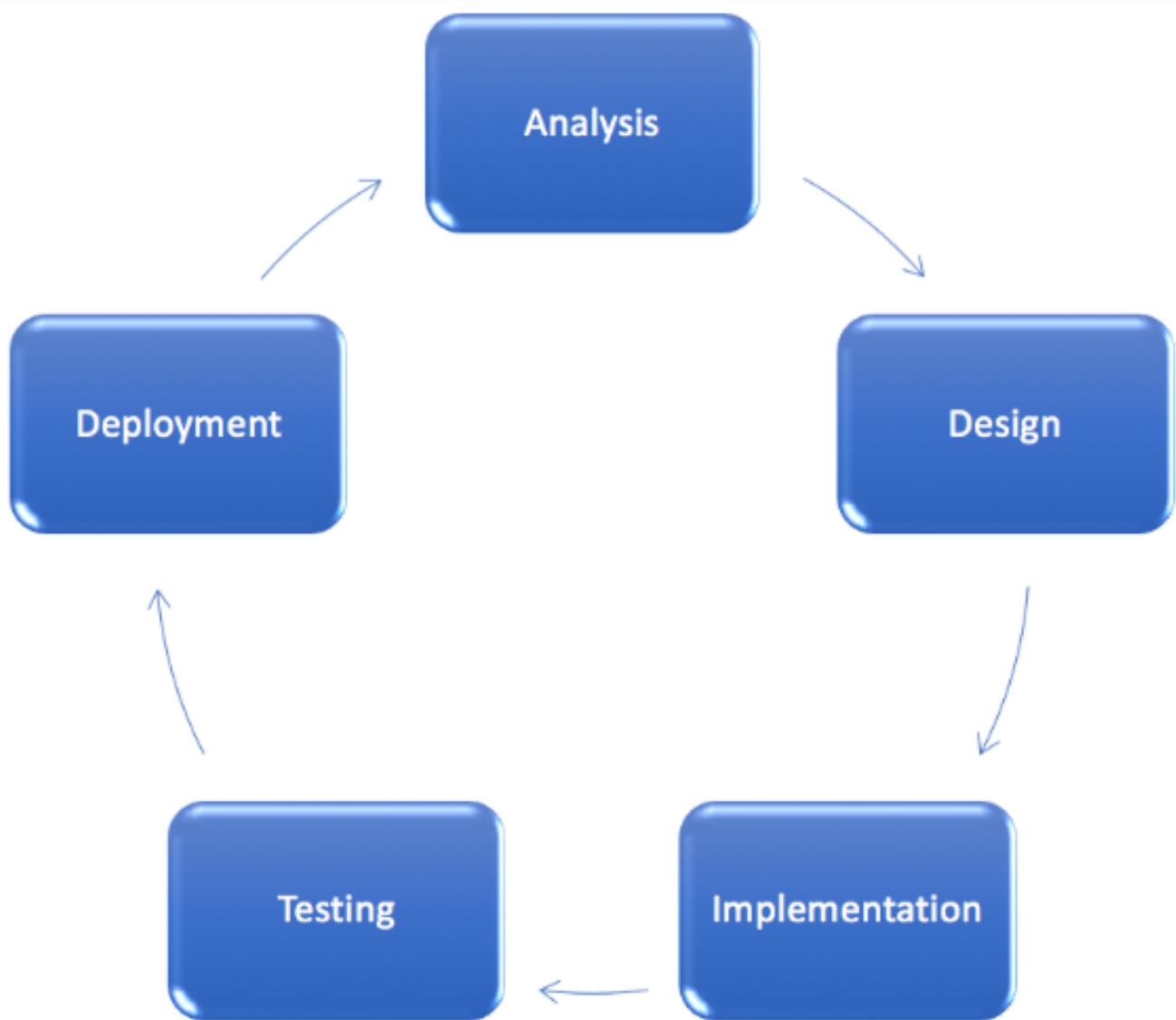
The Software Development Lifecycle, or SDLC, has been referred to by many names; Systems Development Lifecycle, Application Development Lifecycle, etc. These terms all ultimately refer to the same essential concept, one that you should become intimately familiar with; the processes by which software systems are created.

In this lesson, you will explore the SDLC to become more familiar with the overall stages of creating software systems.

What is Lifecycle

For decades, the Software Development Lifecycle has referred to the essential stages of a software product or feature, from concept to delivery. Although the exact outline of these stages will differ depending on who you are talking to, the gist is the same.

A typical description of the SDLC may look something like this:



Analysis

In this stage, stakeholders and domain experts identify what capabilities the system or feature will need.

The goal of this stage is to define **what** functionality is needed in the system.

Based on the methodology that the team adheres to, this stage will typically be run by a Business Analyst or Product Owner.

The outcome of this stage may be comprehensive requirement specifications, or very lightweight user stories depending on how the team operates. Regardless, the need to perform some analysis and determine the requirements of the system is essential to SDLC.

Design

- System Architects or senior technical members will use this stage to evaluate architectural strategies, and available technologies to implement the desired solution. The results of this effort may be a very detailed technical specification, or possibly just a lightweight architectural guideline. Either way, it's essential that the technical team has some guidance on the tooling and patterns that should be leveraged to create a stable, scalable system within the allotted time.
- User Experience (UX) Designers will often leverage this stage to create system mocks, or illustrations showing how the system should look and behave. They may focus group customers, and produce design artifacts such as style guides and pattern libraries. These documents will help guide the User Interface (UI) developers towards creating a cohesive look and feel which satisfies customers and stakeholders.

In the design stage, the team will set out to take the business requirements and transcribe them into both visual and architectural designs for how the system should be created.

The goal of this stage is to define **how** the system should be created.

Implementation

In the Implementation stage, the team will build the system while adhering to the design as closely as possible. This is often the stage which is most prone to delays since the implementation of designed features often uncovers challenges.

Typically, the majority of the team is focused on the implementation phase, where developers and engineers work to bring the designs to life. Considered to be the most complex stage, which bears the most risk, the Implementation phase is also the longest. To reduce the length of this phase, teams will often staff more personnel in the development tasks.

The Implementation stage produces the core functionality of the product or system, which meets the requirements and designs set forth by stakeholders, architects and designers.

Testing

The Testing stage is where Quality Assurance members will ensure that the system is stable and meets specifications. This serves as a detailed check to ensure that the system not only meets the requirements, but is also stable and performs well on intended devices.

Results of the testing phase will inevitably produce bugs and defects that will need to be fixed by the development team. Most teams are aware that defects are a necessary part of the process, but strive to measure and reduce the occurrence.

The testing stage is also high-risk and prone to delays since it often uncovers significant problems within the system that were unanticipated.

The testing stage will often produce QA-related artifacts such as test plans and testing automation systems.

Deployment

In the Deployment stage, the system is prepared for launch to the intended users. Frequently teams will have a dedicated person or group of people assigned to the deployment and maintenance of the system.

Depending on the type of application being built (e.g., web application, mobile app, embedded system, etc.), the deployment process and intended outcome can vary significantly. Regardless, you can typically anticipate that there is a sequence of moving the codebase through multiple environments for verification and user access.

The practice of automating these processes has become increasingly more valuable as companies seek to deploy more rapidly to customers. In many companies, you can now find Development Operations, or DevOps, specialists, tasked with maintaining a Continuous Deployment process.

SDLC and Development Methodologies

As you continue through this course, you will explore some methodologies for managing development processes. It's important to note that the SDLC is not a development methodology so much as an observation of the common stages in software development. You will soon see some more opinionated guidelines for managing software teams; however, most methodologies are recommendations for optimizing how to navigate these essential stages.

KeyTerms

SDLC	Software Development Lifecycle.
Analysis	Identifying the needs of the system or feature.
Design	Creating both the visual and architectural design of the system.
Implementation	Development of the core functionality of the system.
Testing	Verification and Quality Assurance of the system.
Deployment	The process of deploying the system to its intended users.

Software Testing Life Cycle

When software development lifecycle models are selected, the context of the project and product characteristics must be taken into consideration. An appropriate software development lifecycle model should be selected and adapted based on the project goal, the type of product being developed, as well as business priorities. We follow the Software Testing Life Cycle (STLC) to test software and ensure that quality standards are met. Software testers utilize this method where tests are carried out systematically over several phases. During product development, phases of the STLC may be performed multiple times until a product is deemed suitable for organizational release. In this section, we will explore the Phases and Activities of STLC. We will also explore the testing strategy that can help you efficiently meet software quality standards.

usefull

<https://opentextbook.site/informationsystems2019/>

read

<https://opentextbook.site/informationsystems2019/chapter/chapter-10-information-systems-development/>

Phases and activities

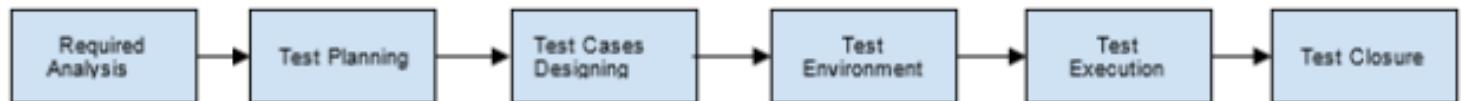
STLC, which is known as Software Testing Life Cycle. STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product. The Certified Tester Foundation Level ISTQB® (a type of software testing certification) specifies seven principles of software testing. Th

These seven principles as stated are:

- Exhaustive testing is impossible
- Early testing saves time and money
- Defects cluster together
- Beware of the pesticide paradox
- Testing is context-dependent
- Absence-of-errors is a fallacy

Test levels are groups of test activities organized and managed together. Each test level is an instance of the test process. These activities are performed at a given point in development. A testing environment consists of elements supporting test execution. The test environment must mimic the production environment to uncover any bugs. Below is a depiction of the test activities.

Keyword	Description
Requirement Analysis	The testing team starts high-level testing concerning the AUT (Application under Test).
Test Planning	Test Team plans the strategy and approach.
Test Case Designing	Develop the test cases based on scope and criteria.
Test Environment Setup	When the integrated environment is ready to validate the product
Test Execution	Real-time validation of product and finding bugs.
Test Closure	Once testing is completed, matrix, reports, results are documented.



Software dev lifecycle

In order for you to understand your role as a software tester, you need to develop an understanding of the fundamental principles and processes of software testing. Upon completion of this module, you will learn enough to have meaningful conversation around software development processes and the roles of those involved in each process, including you the software tester. Prior to any software testing, the software project must be initiated based on organizational requirements. Organizational system requirements are characteristics or features that must be included in an information system to satisfy business requirements and are acceptable to users. Testing is implemented to ensure that system requirements are met.

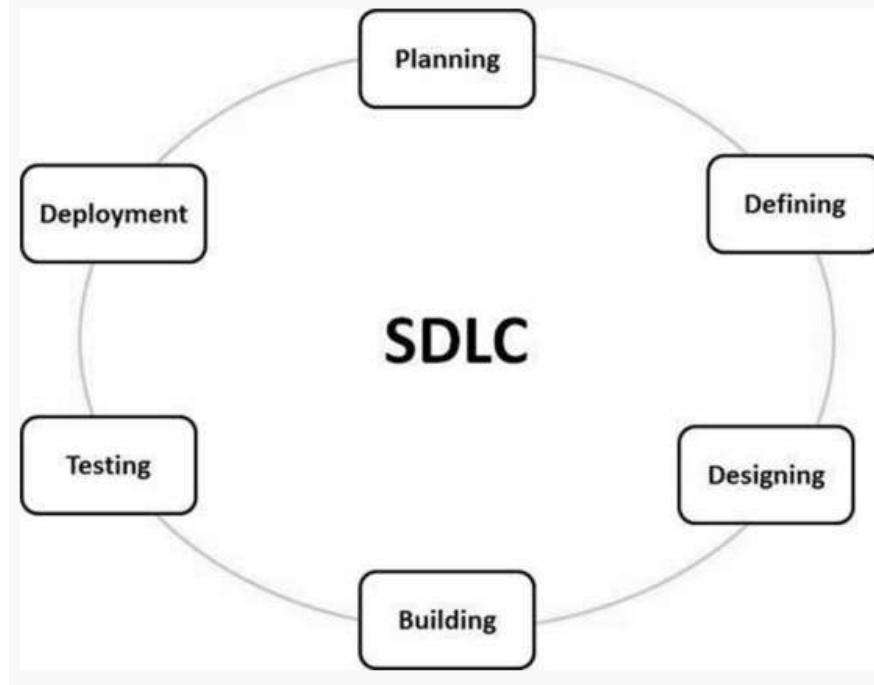
The SDLC or system development life cycle provides the framework in which the software development team operates. We use the Systems Development Life Cycle (SDLC) to manage information systems implementations. As a software tester, you will be working with a systems analyst whose responsibilities include turning the business requirements into an information system. A systems analyst further helps plan, develop, and maintain information systems. Systems analysts must be excellent communicators with strong analytical and critical thinking skills, you will work with them in creating test cases from which you will use in the testing phase.

<https://opentextbook.site/informationsystems2019/>

<https://opentextbook.site/informationsystems2019/chapter/chapter-10-information-systems-development/>

stages of sdlc

System requirements are the foundation of the systems analysis phase within an information systems implementation. It is imperative that each phase within the SDLC is carefully approached with regard to the quality of the system. Problems with the requirements have a ripple effect; negatively affecting all phases of the SDLC. Based on various research studies in systems engineering, we know that poor requirements are a leading cause of failed projects. In this lesson, we delve deep into the phases of the system development life cycle.



keynote

Preliminary Analysis	The study is to characterize the different technical approaches that can be taken to execute the project effectively with the least amount of risks
System Analysis	Define and document the product requirements seeking approval from the client
System Design	The best architecture or platform for which the product should be developed
Programming	To follow the coding guidelines defined by their organization and programming tools to generate the code needed for the project
Test	When the code is tested to make sure it runs properly
Deploy	Released to the market
Maintenance	Support process in place just in case there are bugs to be fixed

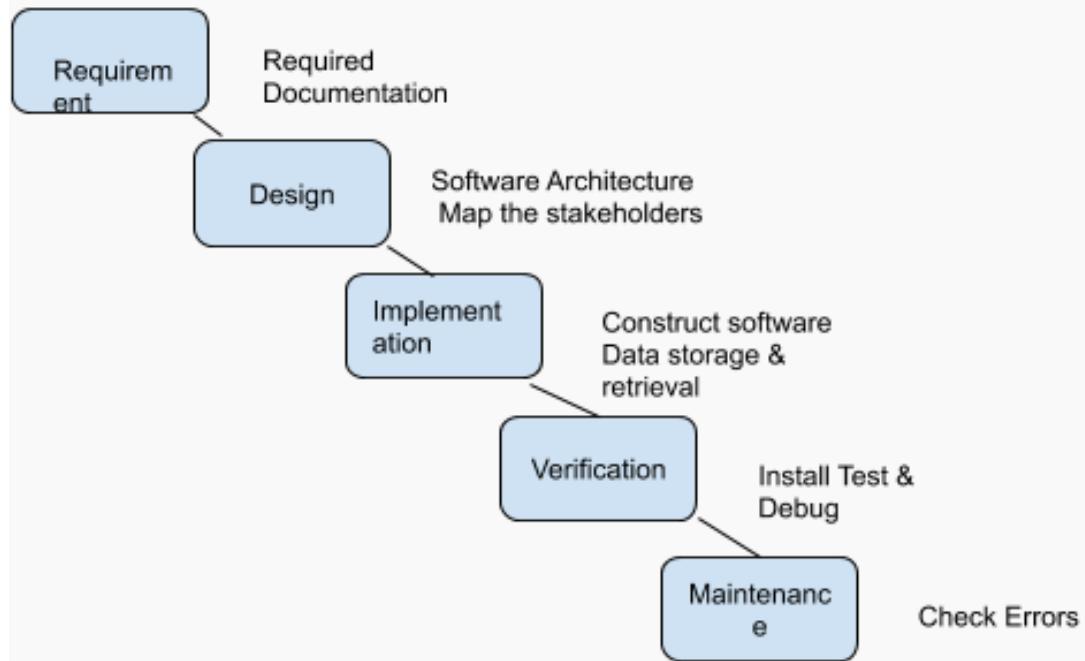
SDLC models

Even though the SLDC has six distinctive stages, there are several methodologies (approaches) used to guide the software development team into systems implementation. Each process model follows a series of steps unique to its type to ensure success in the process of software development. Examples of the approaches are iterative model, waterfall model, spiral model, V model, big bang model, and Agile. No matter which software development lifecycle model is chosen, testing activities must start in the early stages of the systems development lifecycle. Let's explore both sequential and Iterative and incremental development models to software development.

Note: A sequential development model describes the software development process as a linear, sequential flow of activities. Using this approach project teams deliver software that contains the complete set of requirement features, however, testing can take a long time and the final delivery can take a long time to deliver to stakeholders.

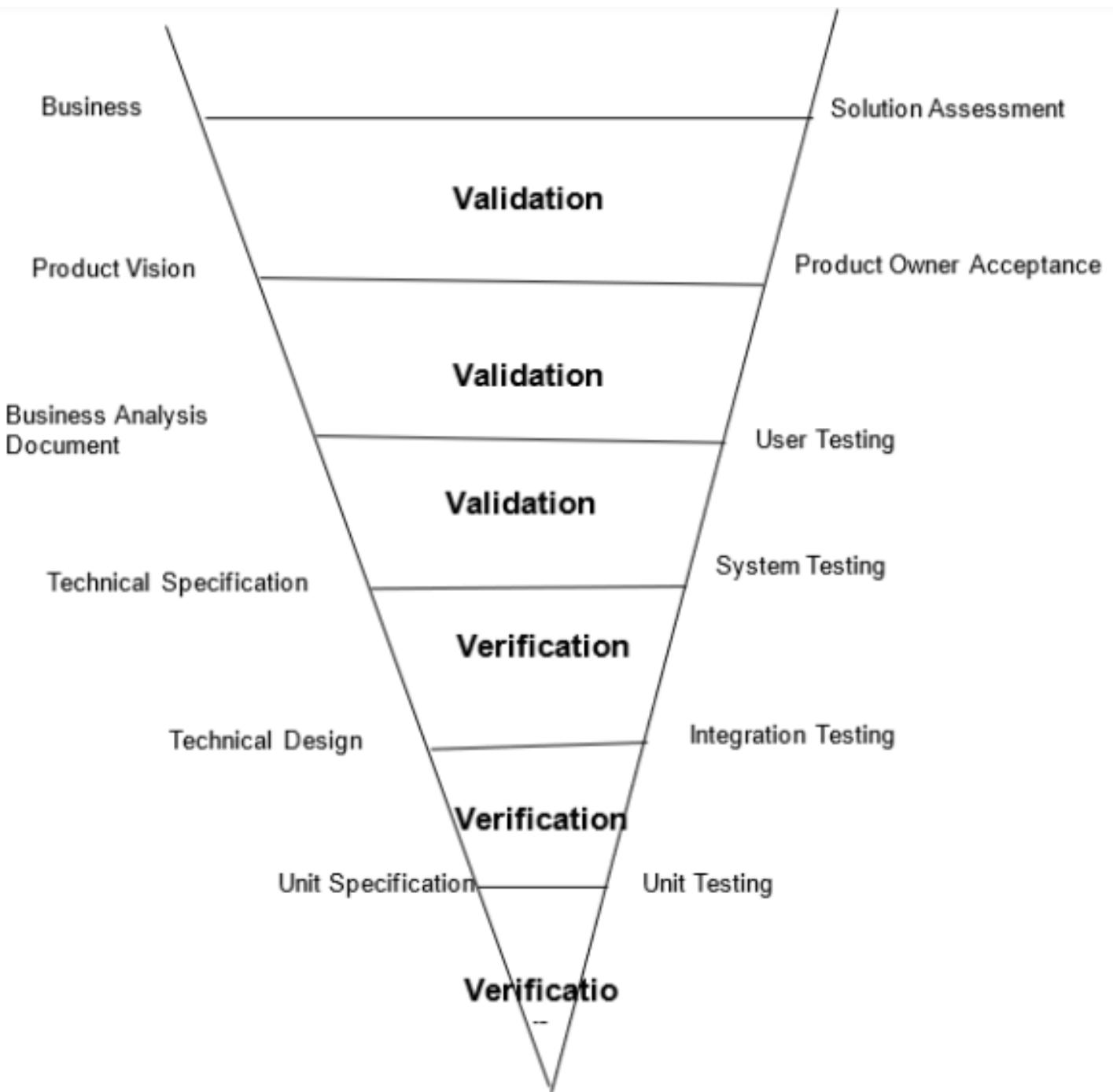
water model

The waterfall approach is one of the most established and oldest SDLC models used within business. Using the waterfall model, the approach to development is a simple one, yet a disadvantage is its lack of flexibility in changing requirements due the rigid structure that demands all system requirements be defined at the very start of a project's implementation. In systems implementation, many times flexibility is needed especially in situations where the requirements may often change. In the waterfall model, test activities only occur after all other development activities have been completed.



V Models

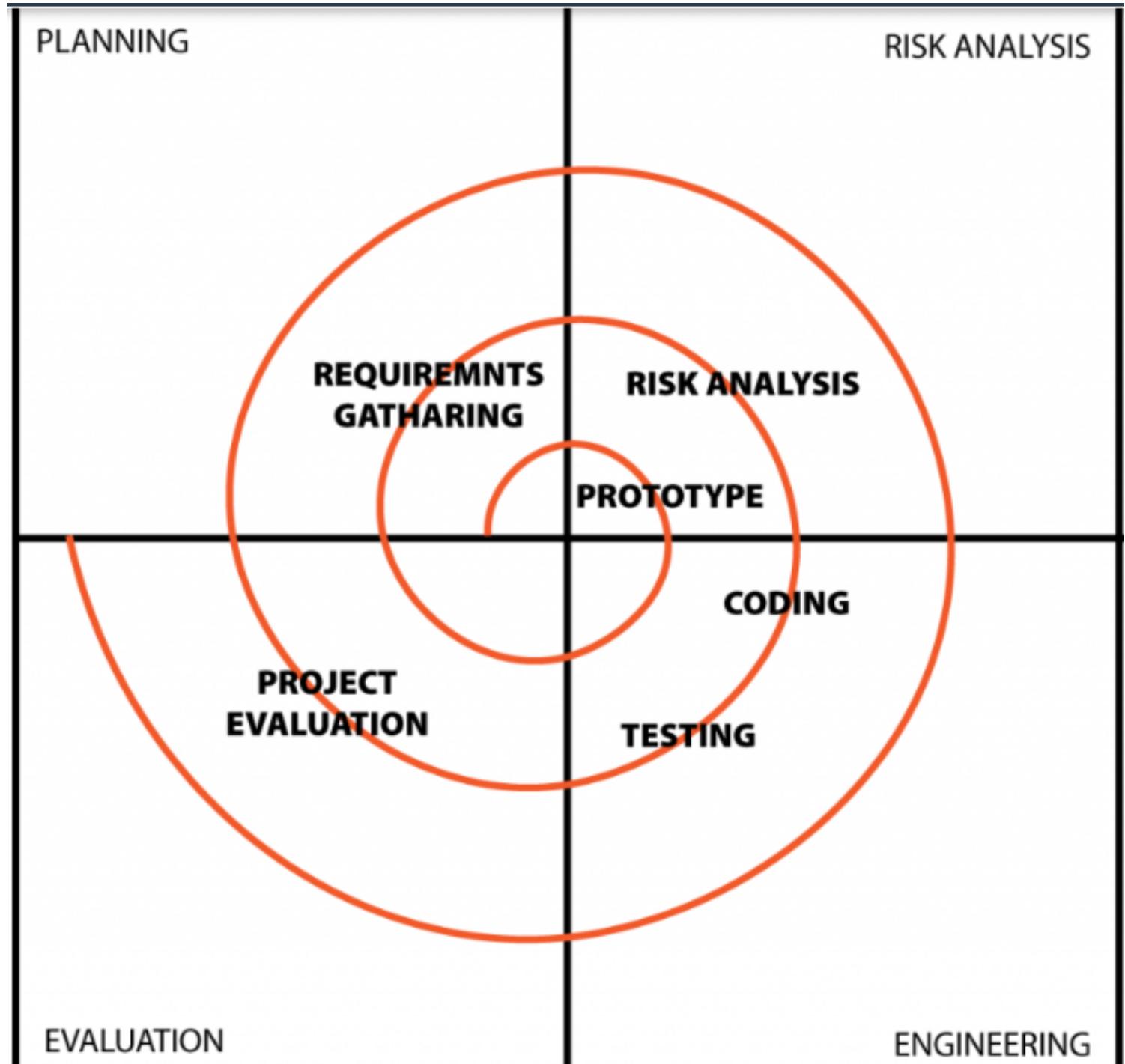
In the v-model, we approach the waterfall methodology in a similar way that we do the waterfall methodology. However, in the coding phase, the process steps are flipped up after coding. The v-model has a very strict approach. The next phase begins only when the previous project phase is complete. An advantage of the V Model is it works very well for smaller projects. One disadvantage is once an application is in the testing stage, it is difficult to go back and change the functionality. Unlike the Waterfall model, using the V-model integrates the test process throughout the development process. In the V model, the execution of tests associated with each test level proceeds sequentially, but in some cases, overlapping occurs.



Iterative and incremental development models

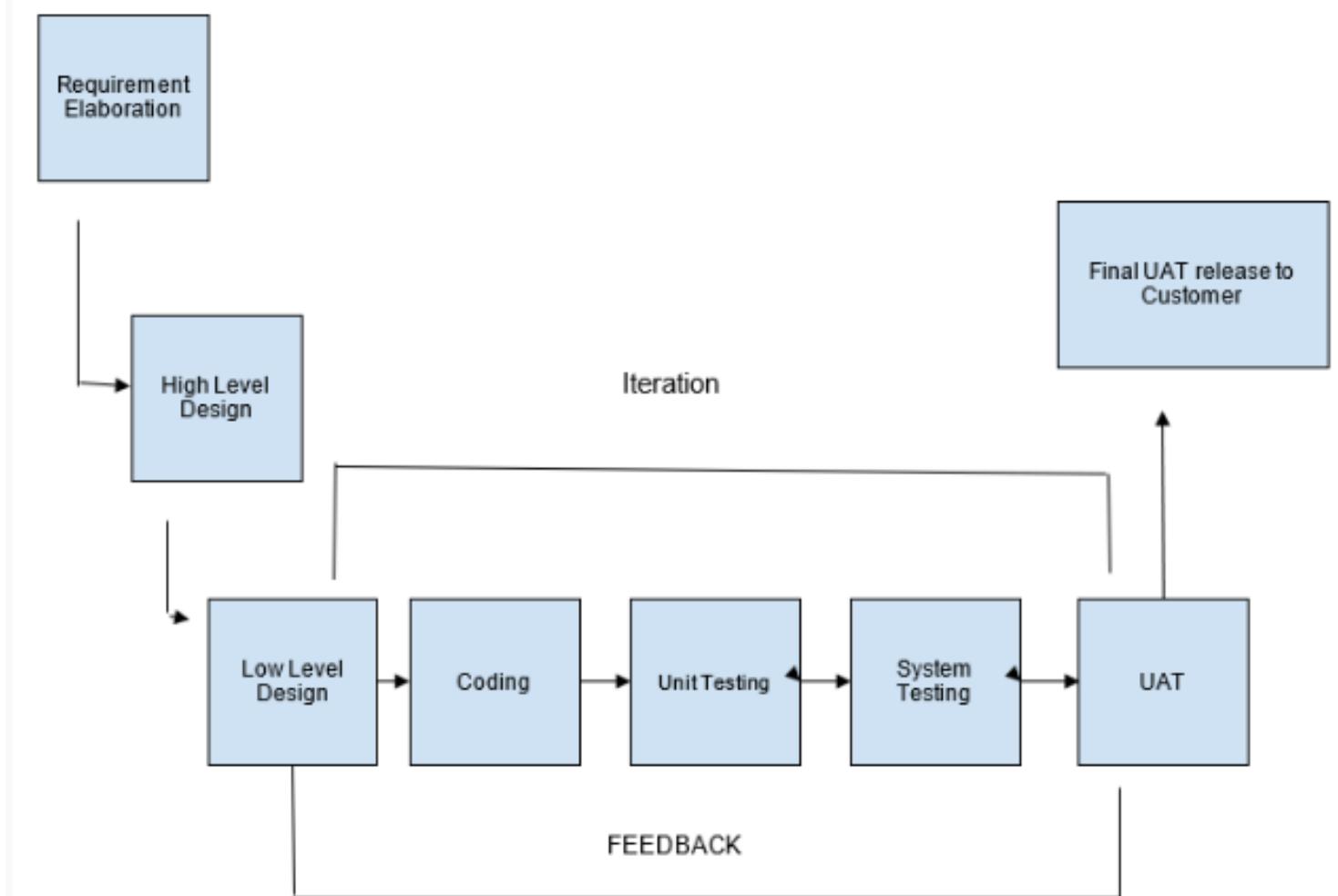
Spiral Model

The Spiral Model is a combination of the waterfall model and an iterative model. One advantage of the Spiral model is that it allows space for customer feedback and one disadvantage is not recommended for smaller projects because it is not cost-effective.



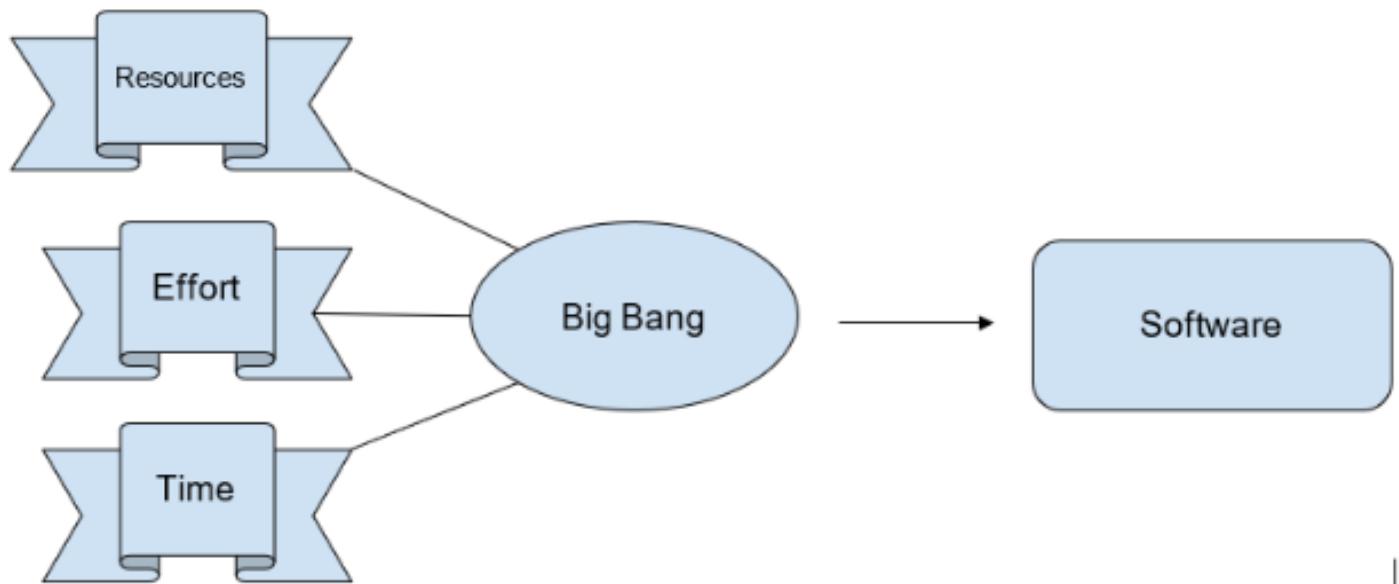
Iterative Model

In implementing the iterative model approach, the waterfall model is followed, yet extended as it cycles through it several times in small increments. Following the iterative approach, the project is broken down into mini-projects. Essentially once the requirements are identified, the review is extended to identify additional requirements iteratively. Following the iterative model, the requirement process is repeated, producing a new version of the software at the end of each iteration. Some advantages are that risk management is easier, and it is less expensive.



Big Bang Model

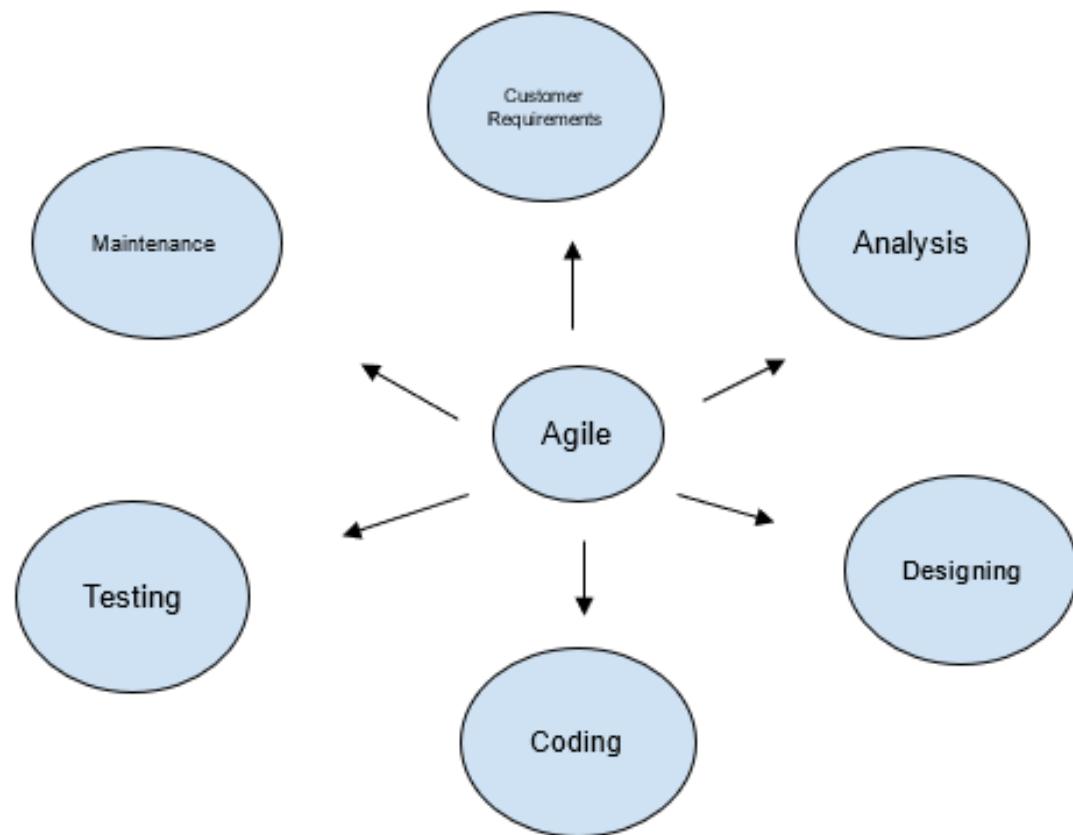
The Big Bang model is an approach where no specific process is followed. The Big Bang Model does not follow a process/procedure, and there is very little planning required. One advantage of the Big Bang model is it is very easy to manage and very flexible. A disadvantage is that it is not a reliable model for complex long ongoing projects.



|

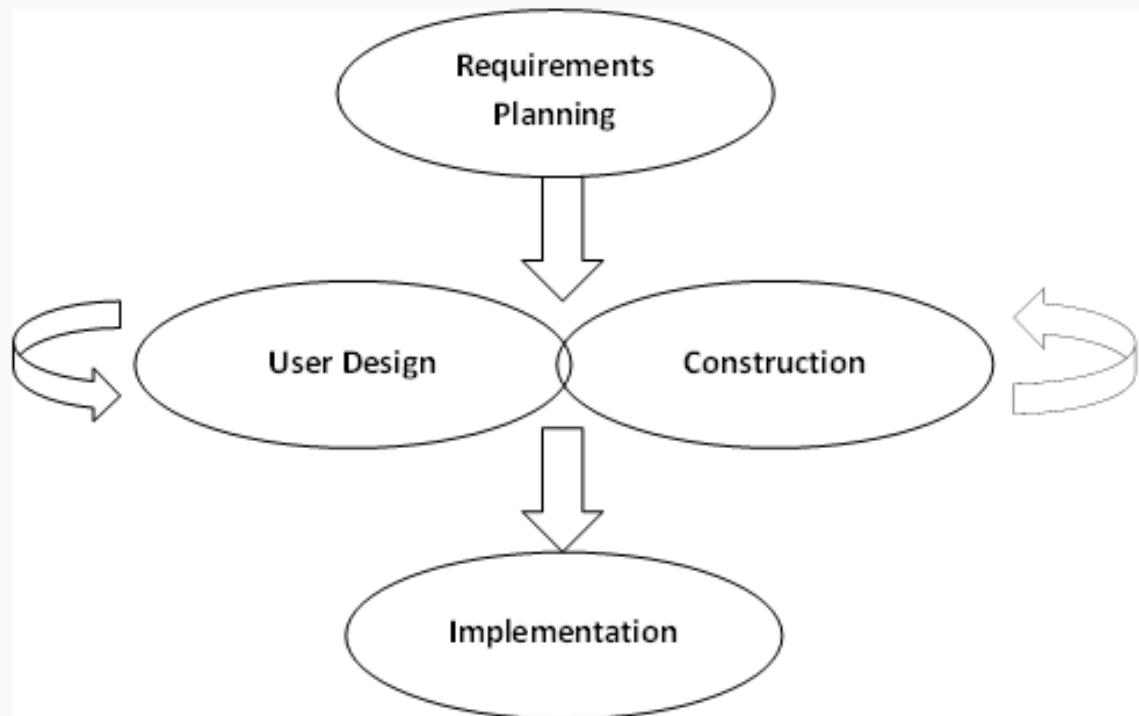
Agile

Using the Agile approach, the tasks are divided into small time frames to deliver specific features for a release. Some advantages of this model are that it promotes teamwork and cross-training within the developers. Some disadvantages are if the customer is not clear on what they are looking for, that might cause the developers to go in the wrong directions, and it's not suitable for handling complex projects.



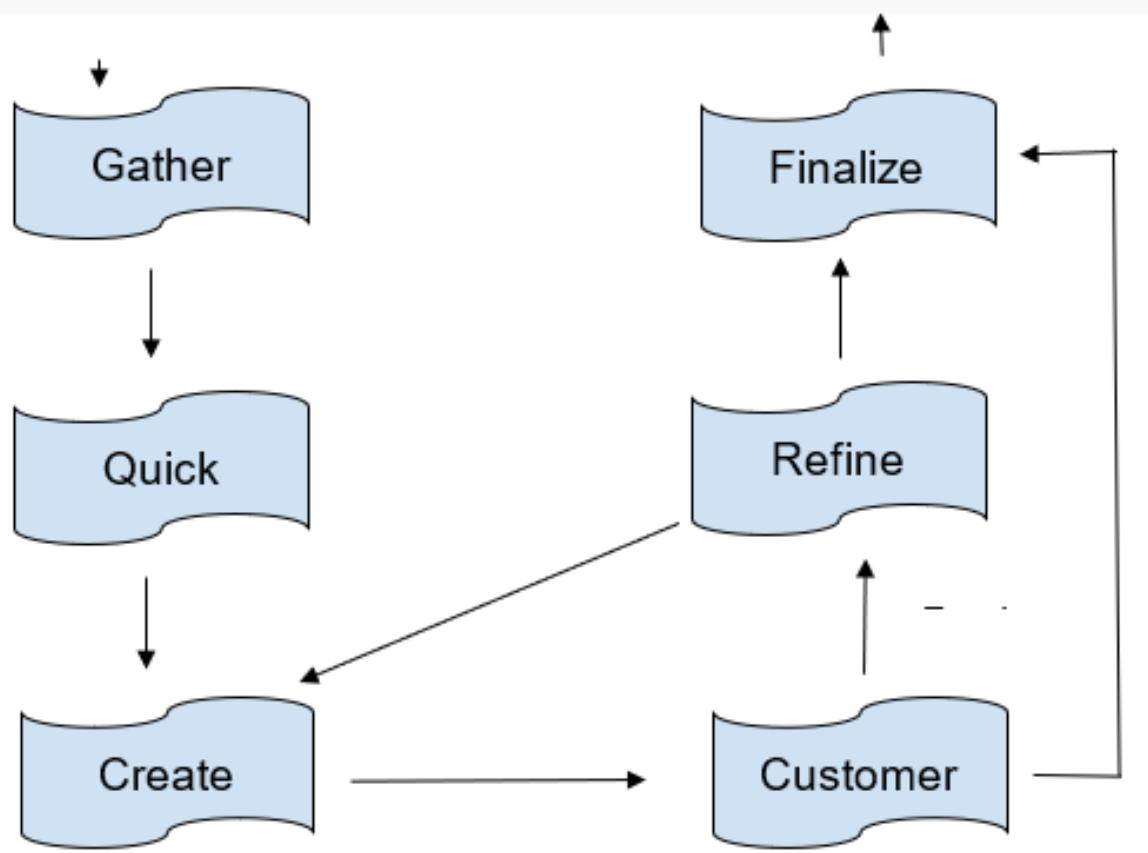
RAD Model

The RAD (Rapid Application Development) model uses focus groups, early testing of the product, and workshops for obtaining the information about what the customer wants, then reuse the prototype components for continuous integration and rapid delivery. Some advantages are shorter development time, and it encourages the client's feedback. Some disadvantages are requiring some very skilled developers and the cost is extremely high. Projects that utilize RAD implement iterative and incremental approaches. Overall software testing time is greatly reduced in the RAD model as prototypes are independently tested during each and every iteration.



Software Prototype

The Software Prototyping approach refers to the building of application prototypes. These prototypes display the functionality of the product under development, however, may not hold the exact logic of the original software requirements. In the software prototypes model, the team works to understand customer requirements at an early stage of development. Having customer feedback at the early stage, the developer knows exactly what type of product the customer wants. Some advantages are very easy to detect errors, and it is perfect for an online system. Some disadvantages are that it's very expensive and may increase the complexity of the system.



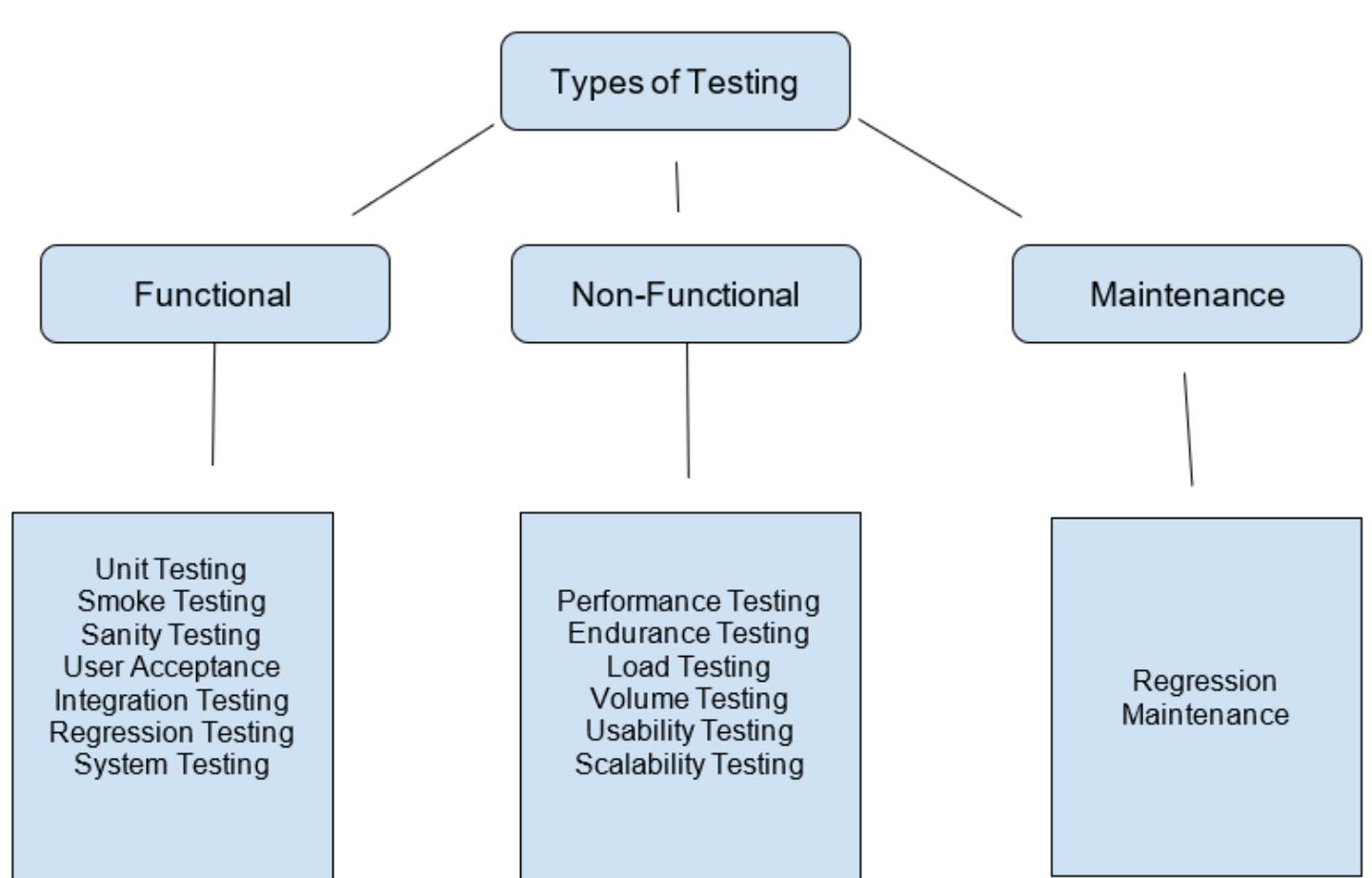
Summary

Software development is the systematic process of mastering what the customer needs, so you can develop software to solve their business problems. SDLC provides a framework for certain activities during the seven stages of development which are:

1. Required information from the client
2. Feasibility study
3. Design
4. Coding
5. Testing
6. Deployment
7. Maintenance

testing

The most important component of testing is finding bugs as early as possible, so they can be fixed and to make sure it is working as expected.



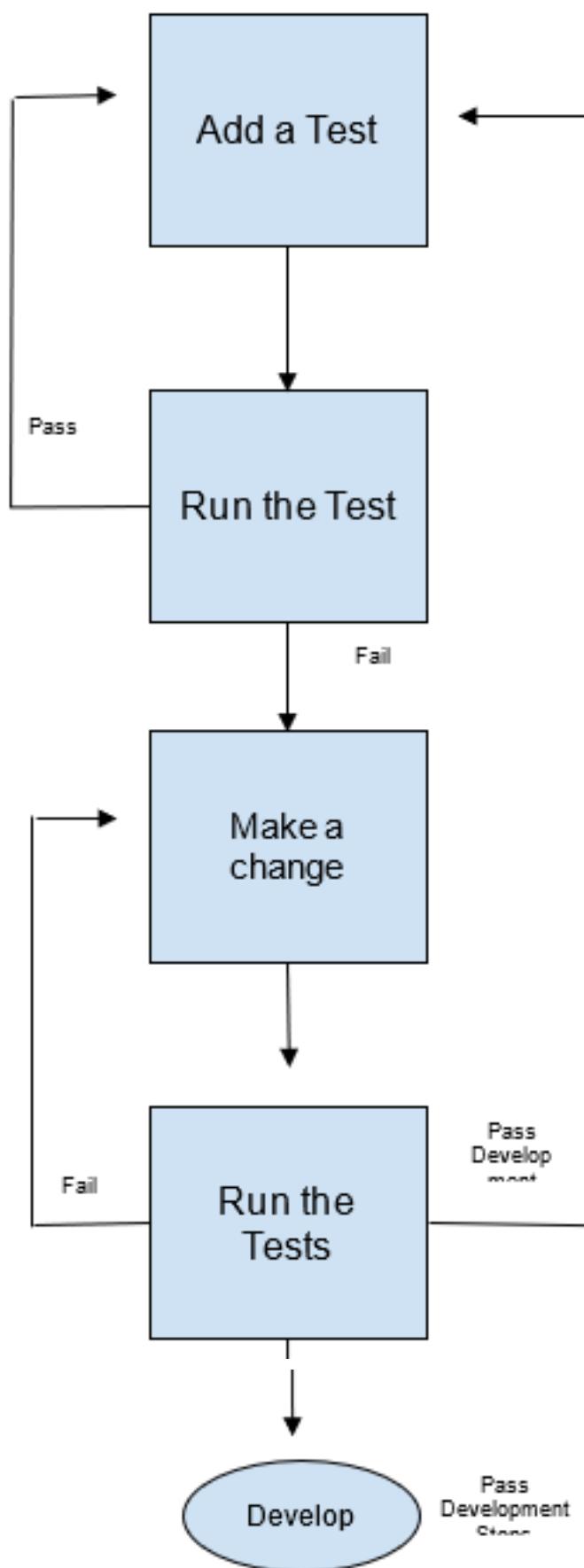
Test Driven Development

Test Driven Development (TDD) is a software development approach in which test cases are created to indicate and authenticate what the code will do. The code is created and tested if for some reason it fails, or it has bugs then a new code is written to make sure it is bug-free.

Myths and misconceptions

TDD is too Time Consuming. The Business Team Would Never Approve	Your business does not care just as long as it's effective
You cannot design tests until you know the design, and you can't know the design until you implement it.	Developers often start implementation before they know what the API will look like.
Script all tests before you start the code	<ol style="list-style-type: none">1. Write one test2. Watch it fail3. Implement the code4. Watch the test pass5. Repeat
Red, Green, and ALWAYS Refactor	Review code for improvement opportunities.
Everything needs Unit Test	Unit tests work best for pure functions

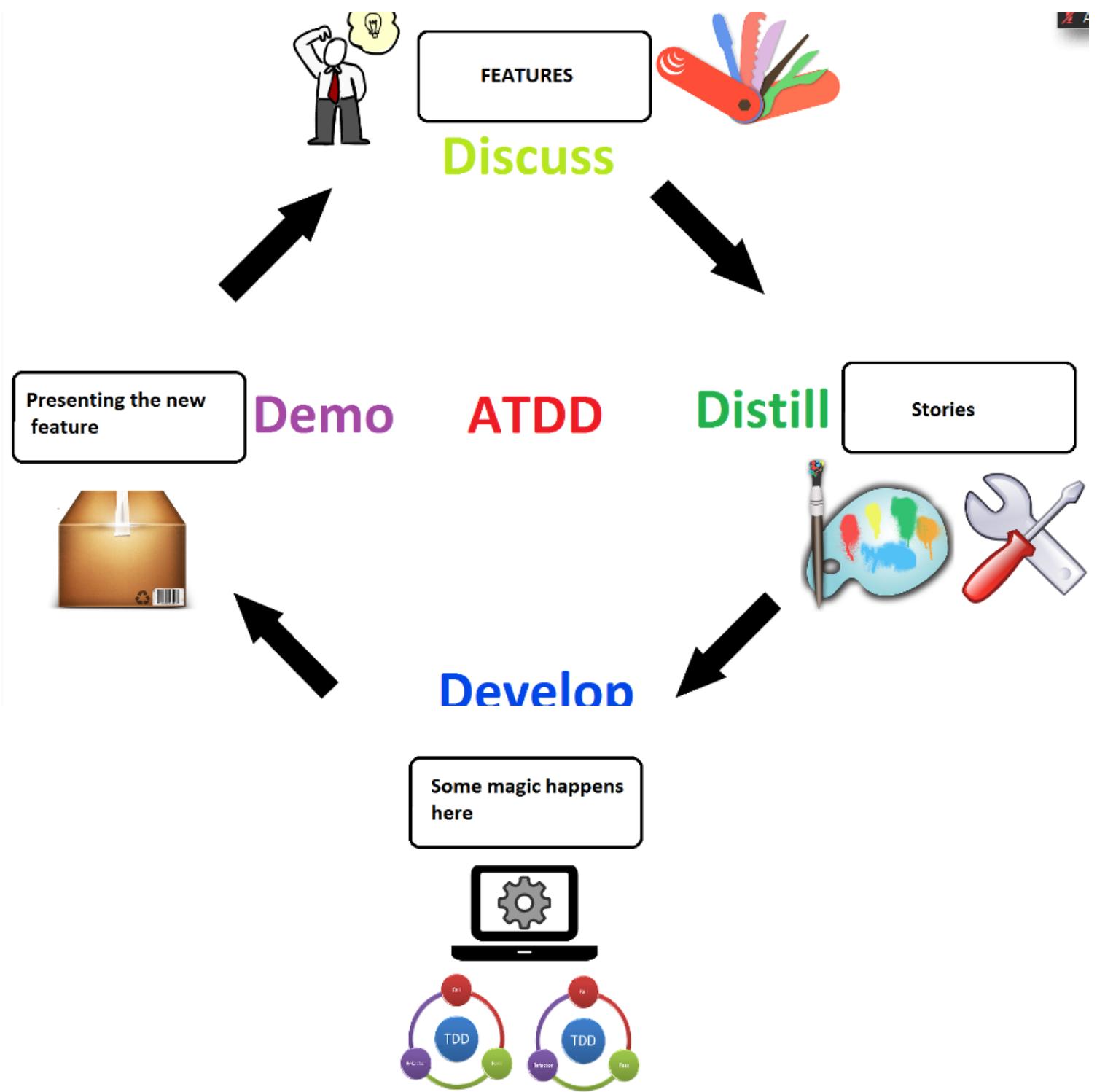
Performing a Unit Test



Acceptance Test Driven Development

Acceptance Test-Driven Development (ATDD) a technique that answers the question "does the code work as expected"

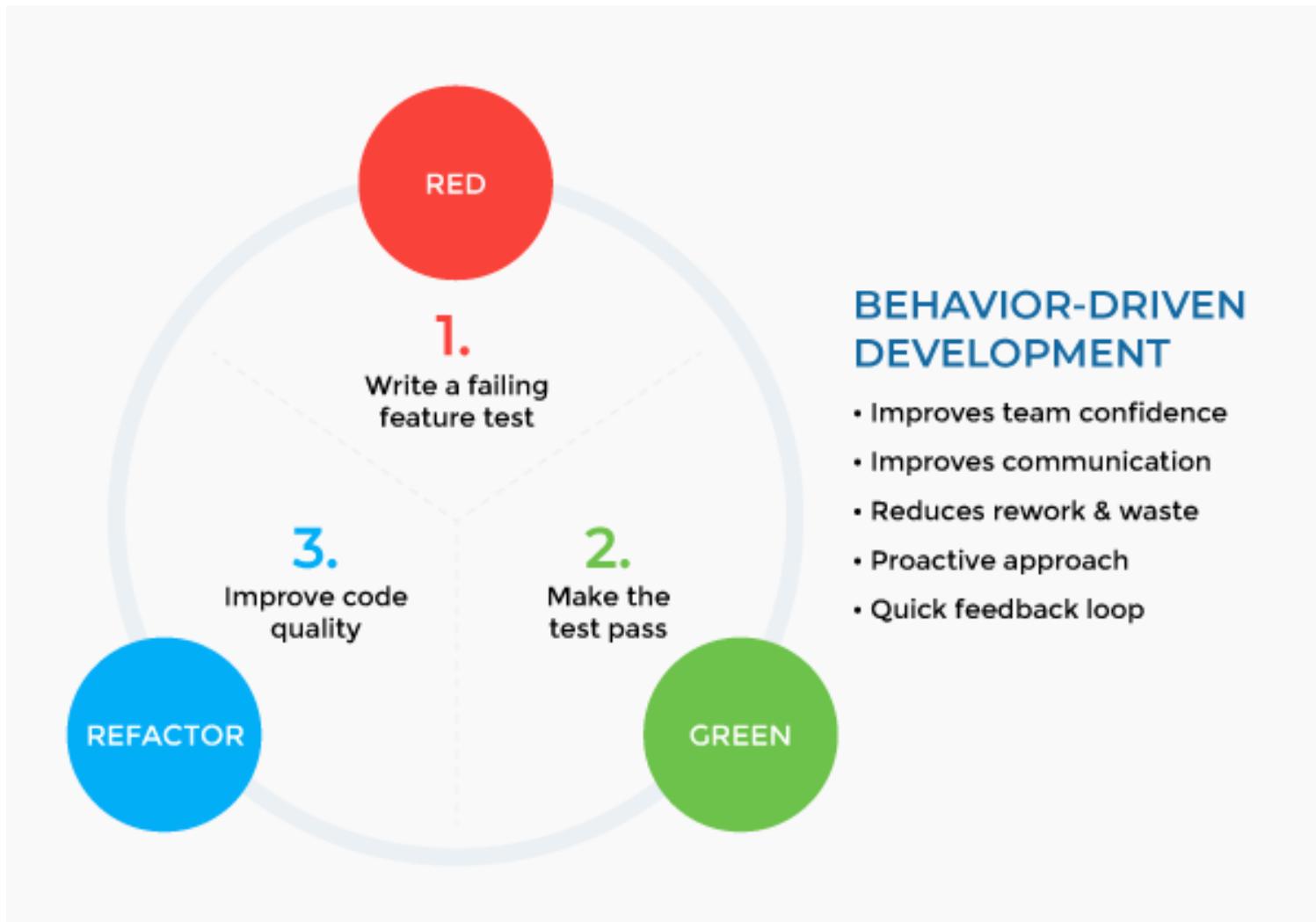
example



Behavior Driven Development

Behavior Driven Development (BDD) uses examples in illustrating the behavior of the system.

example



Test documentation	Helps to estimate the testing effort needed, test coverage, resource tracking, execution progress, etc. It allows you to describe and document test planning, test design, test execution, test results that are drawn from the testing activity
Functional testing	Validates the software system against the functional requirements/specifications.
Nonfunctional testing	Validates the software system against the functional requirements/specifications.

java

abstract

Abstract classes are classes which can contain methods akin to those found in non-abstract classes. This means a regular class can be turned into an abstract class. They contain the `abstract` keyword preceding the `class` keyword in a class declaration. Below is a class which has been converted into an `abstract` class simply by adding the `abstract` keyword.

```
public abstract class Person {  
    private String name;  
    private int age;  
  
    public Person(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
  
    public void sayHello(){  
        System.out.println("Hello, my name is " + this.name);  
    }  
  
    public int getAge(){  
        return this.age;  
    }  
}
```

abstract vs interface

There are a few things to note when differentiating abstract and interface classes.

1. The keyword `interface` is used to declare interfaces, while the keyword `abstract` is used to define abstract classes.
2. Interfaces can only contain abstract methods, while abstract classes can contain a mix of abstract and non-abstract methods. Although the methods in the interfaces don't explicitly have `abstract` prepended to them, all methods in interfaces are abstract.
3. Interfaces can inherit from multiple classes (multiple-inheritance), while abstract cannot.
4. Interfaces cannot provide implementation, while an abstract class can.
5. Interfaces can only contain final and static variables, while an abstract can contain final, non-final, static, and non-static variables.

exp

Person class:

```
public abstract class Person {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public abstract void sayHello();  
  
    public String getName() {  
        return this.name;  
    }  
}
```

Teacher class:

```
public class Teacher extends Person {  
    private String subject;  
  
    public Teacher(String name, int age, String subject) {  
        super(name, age);  
        setSubject(subject);  
    }  
  
    public void setSubject(String subject) {  
        this.subject = subject;  
    }  
  
    public void sayHello() {  
        System.out.println("Hello, my name is " + this.getName() + ", and I teach " + this.subject);  
    }  
}
```

call the function

```
Teacher teacher = new Teacher("Tim Boyd", 35, "English");  
teacher.sayHello();
```

bufferedReader

`BufferedReader` is a class that makes reading and writing files more efficient.

`FileReader` is a powerful tool on its own. However, each time a character is being read from a file, the operating system needs to wait for other programs to finish performing operations on the disk, and then retrieve the single character from the file. I/O requests take quite a long time compared to memory (RAM) and CPU time.

`BufferedReader` makes the trip to the hard disk worthwhile and efficient by grabbing more data (e.g., a full line of text) for every trip. The same is true of `BufferedWriter` being a more efficient alternative to writing to a file than `FileReader`. However, this is not to say that one needs to be used over the other. They can be used together in series.

example

Objects can be passed as parameters to other methods and constructors. The `BufferedReader` constructor takes in a `FileReader` object. The following code snippet demonstrates what this would look like:

```
FileReader fileReader = new FileReader("story.txt");
BufferedReader reader = new BufferedReader(fileReader);
```

The above can be shortened to something like the following:

```
BufferedReader reader = new BufferedReader(new FileReader("story.txt"));
```

exam

```
// define the variables and objects
BufferedReader reader;
FileWriter writer;
String fileContentsRead = "";

// attempt to run code that may cause an error
try{
    // open the file for writing
    writer = new FileWriter("story.txt");
    // write the letter "a" inside of the file
    writer.write("a");
    // close the file (finished writing)
    writer.close();

    // open the file for reading
    reader = new BufferedReader(new FileReader("story.txt"));
    String line;
    while ((line = reader.readLine()) != null) {
        fileContentsRead = fileContentsRead + line;
    }
    // close the file (finished reading)
    reader.close();
}
```

```
}

// catch errors that happened in the try block
catch(Exception e){
    // print the stack trace (error)
    e.printStackTrace();
}

finally{
    System.out.println(fileContentsRead);
}
```



built-in-classes

exception_handling

```
public static void main(String[] args) throws Exception {  
    // SSN is 123-45-6789  
    int SSN = 123456789;  
  
    try {  
        // if SSN is 000-00-0000  
        if(SSN == 000000000){  
            throw new Exception("User SSN Undeclared!");  
        }  
        else {  
            System.out.println("Everything looks good here");  
        }  
    }  
    catch(Exception e) {  
        System.out.println(e.getMessage());  
    }  
}  
  
int result = 0;  
  
try{  
    result = 1/0;  
}  
catch(Exception e){  
    System.out.println("An error occurred when calculating the result.");  
    result = -1;  
    e.printStackTrace();  
}  
finally{  
    System.out.println("The new result is: " + result);  
}
```

file_i-o

```
1 package com.jamesgosling.helloworld;
2
3 import java.io.FileReader;
4
5 public class HelloWorld {
6     public static void main(String[] args) throws Exception {
7         FileReader inputStream;
8         FileWriter outputStream;
9
10        // FileWriter cannot be resolved to a type
11        // 16 quick fixes available:
12        //   - Import 'FileWriter' (java.io)
13        //   - Create class 'FileWriter'
14        //   - Create interface 'FileWriter'
15        //   - Change to 'FileFilter' (java.io)
16        //   - Change to 'FileFilter' (javax.swing.filechooser)
17        //   - Change to 'FilenameFilter' (java.io)
18    }
19 }
```

example

```
// define the variables and objects
FileReader reader;
FileWriter writer;
String fileContentsRead = "";

// attempt to run code that may cause an error
try{
    // open the file for writing
    writer = new FileWriter("story.txt");
    // write the letter "a" inside of the file
    writer.write("a");
    // close the file (finished writing)
    writer.close();
    
    // open the file for reading
    reader = new FileReader("story.txt");
    // save each character as an integer (ASCII)
    int c;
    // while there are more characters to be read, store them
    while ((c = reader.read()) != -1) {
        fileContentsRead = fileContentsRead + c;
    }
    // close the file (finished reading)
    reader.close();
}

// catch errors that happened in the try block
catch(Exception e){
    // print the stack trace (error)
    e.printStackTrace();
}

finally{
    System.out.println(fileContentsRead);
}
```

Hashtable

A dictionary, also called a *hash table*, functions like a dictionary in the real world. Each word acts as the *key* which is associated with the definition, or the *value*. In Java, the key and value of a dictionary can be any data type, not just a String as seen in real world dictionaries. A dictionary for the English language would consist of a key of type `String` and a value of type `String`. If this were written in code, it would resemble the following code block.

Since the built-in `Hashtable` class is being used, be sure to import the library at the top of the class file near the package statement: `import java.util.Hashtable;`.

```
// <Key, Value>
Hashtable<String, String> englishDictionary = new Hashtable<String, String>();
```

example

```
Hashtable<String, String> englishDictionary = new Hashtable<String, String>();
englishDictionary.put("persist", "to go on resolutely or stubbornly in spite of opposition, importunity, or warning");

Hashtable<String, String> englishDictionary = new Hashtable<String, String>();
englishDictionary.put("persist", "to go on resolutely or stubbornly in spite of opposition, importunity, or warning");

// get the value associated with the "persist" key
String definition = englishDictionary.get("persist");
// if the key exists, print the value
if (definition != null) {
    System.out.println("persist = " + definition);
}
```

stringbuilder

- **StringBuilder(CharSequence cs):** The constructor will initialize the StringBuilder object with the passed in CharSequence (i.e., String-like) and pad on an extra 16 spaces for more elements.
- **StringBuilder(String s):** The StringBuilder object is initialized with the passed-in string with 16 extra spaces for elements appended to the end.
- **StringBuilder(int initialCapacity):** The default size for storing characters is 16. By passing in an integer, that value will be used to declare enough space in the computer's memory to store some number of spaces.

```
public static void main(String[] args) {  
    String cities[] = {"Tokyo", "Delhi", "Shanghai", "Mexico City", "S  
ão Paulo";  
    StringBuilder sb = new StringBuilder();  
    // loop through all of the cities  
    for(int i = 0; i < cities.length; i++){  
        // concatenate the city names  
        sb.append(cities[i]);  
        // concatenate a comma between each city name for increased re  
adability  
        sb.append(", ");  
    }  
    System.out.println(sb);  
}
```

collections

A **Collection** is an interface in which the object organizes a group of other objects. When a class implements the methods in **Collection**, the class specifies how the elements are managed and accessed. This added layer of abstraction may seem unnecessary or tedious initially. Fortunately, Java provides a set of classes that represent a few different types of collections. The code-exercises in this lesson will help solidify the concepts of **Collections**.

arraylist

The trickle down of interfaces and inheritance stops here. `ArrayList` is a class implementation of the `List` interface, which is an extension of the `Collection` interface. An array shares similarities to `ArrayList` in that indexes can be modified. The `ArrayList` can hold any data type. The benefit that `ArrayList` has over an array is the ability to resize easily.

```
// ArrayList  
List<String> arrayList = new ArrayList<String>(3);
```

example

```
List<String> statesOfMatter = new ArrayList<String>();  
statesOfMatter.add("Solid");  
statesOfMatter.add("Liquid");  
statesOfMatter.add("Gas");  
  
// print the values in the ArrayList  
for(int i = 0; i < statesOfMatter.size(); i++){  
    System.out.println(statesOfMatter.get(i));  
}
```

constructor

A *constructor* is how you create the new objects. When you instantiate a new object, the code in the constructor body is executed. For illustration purposes, now add a print statement in the constructor. The syntax for a constructor is the keyword `public` followed by the class name.

```
public class Student {  
    String courseFocus;  
    String firstName;  
    String lastName;  
  
    // constructor  
    public Student(){  
        System.out.println("Student object created!");  
    }  
}
```

You can go back to the main method and instantiate a new object.

```
public static void main(String[] args) {  
    Student student1 = new Student();  
}
```

```
public class Student {  
    String courseFocus;  
    String firstName;  
    String lastName;  
  
    // constructor  
    public Student(){  
        firstName = "";  
        lastName = "";  
        courseFocus = "";  
    }  
}
```

overloading

```
public class Student {  
    String firstName;  
    String lastName;  
    String courseFocus;  
  
    // constructor  
    public Student(){  
        firstName = "";  
        lastName = "";  
        courseFocus = "";  
    }  
  
    // overloaded constructor  
    public Student(String first, String last, String focus){  
        firstName = first;  
        lastName = last;  
        courseFocus = focus;  
    }  
}
```



```
public static void main(String[] args) {  
    Student student1 = new Student();  
    Student student2 = new Student("James", "Gosling", "Java");  
}
```

generic types

Generics are classes that can manage objects, even when the class type is not known. An example of this is an `ArrayList`, which is designed to operate on any object type, where the object type is placed between the diamond `<>`. `ArrayList` is a powerful tool due to its ability to accommodate to any class. Developers can take advantage of the idea of generics within their own code base. Generic types also improve code stability by reducing runtime errors that can occur during typecasting, or the conversion from one object type to another. As you might imagine, this is a useful characteristic that developers want for their code. Generics call attention to bugs at compile-time before the application has a chance to run: this is another benefit to using an IDE such as Eclipse.

example

```
// parameters: make, color, fuel level, trunk width
Car car = new Car("Honda", "Blue", 10, 555);

// casting a Car object to a
Vehicle object Vehicle vehicle = (Vehicle)car;

public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}

// main
Square square = new Square(4.0f);
Box box = new Box();
box.set(square);
Vehicle vehicle = (Vehicle) box.get();
vehicle.printDetails();
```

getters and setters

The default constructor would create a student object with no name. The overloaded constructor you've created allows you to create a Student object with a first name, last name, and course focus. But, you still can't access the data within the object. How do you get the first name of the student? Methods. There two common types of methods called *getters* and *setters*. A *getter* method will return the value within an object, such as the first name. A *setter* will typically set the value of a variable.

example

```
public class Student {  
    String firstName;  
    String lastName;  
    String courseFocus;  
    private String initials;  
  
    // constructor  
    public Student(){  
        firstName = "";  
        lastName = "";  
        courseFocus = "";  
        initials = "";  
    }  
  
    // overloaded constructor  
    public Student(String first, String last, String focus){  
        firstName = first;  
        lastName = last;  
        courseFocus = focus;  
        setInitials();  
    }  
}
```

```
// getter method
public String getFirstName(){
    return firstName;
}

// setter method
public void setFirstName(String newFirstName){
    firstName = newFirstName;
}

// getter method
public String getLastname(){
    return lastName;
}

// setter method
public void setLastName(String newLastName){
    lastName = newLastName;
}

// getter method
public String getCourseFocus(){
    return courseFocus;
}
```

```
    return courseFocus;
}

// setter method
public void setCourseFocus(String newCourseFocus){
    courseFocus = newCourseFocus;
}

// private setter method
private void setInitials(){
    String firstNameInitial = firstName.substring(0,1);
    String lastNameInitial = lastName.substring(0,1);
    initials = firstNameInitial + lastNameInitial;
}

// getter method
public String getInitials(){
    return initials;
}
}
```

example

```
public class Student {  
    String firstName;  
    String lastName;  
    String courseFocus;  
  
    // constructor  
    public Student(){  
        firstName = "";  
        lastName = "";  
        courseFocus = "";  
    }  
  
    // overloaded constructor  
    public Student(String first, String last, String focus){  
        firstName = first;  
        lastName = last;  
        courseFocus = focus;  
    }  
  
    // getter method  
    public String getFirstName(){  
        return firstName;  
    }  
}
```

```
// setter method
public void setFirstName(String newFirstName){
    firstName = newFirstName;
}

// getter method
public String getLastName(){
    return lastName;
}

// setter method
public void setLastName(String newLastName){
    lastName = newLastName;
}

// getter method
public String getCourseFocus(){
    return courseFocus;
}

// setter method
public void setCourseFocus(String newCourseFocus){
    courseFocus = newCourseFocus;
}

}

public static void main(String[] args) {
    Student student1 = new Student();
    Student student2 = new Student("James", "Gosling", "Java");
    String firstNameStudent2 = student2.getFirstName();
    System.out.println(firstNameStudent2);
}
```

static classes

You've seen classes that require you to create an object to be useful. However, there is another type of class, known as a *static class*, which does not need an object instance to be used. If a class does not need to keep track of state, it can be made static. This has already proven to be useful when printing to the console, `System.out.println()`, and launching the Java application, `public static void main()`.

Static classes have the keyword `static` appended after the access modifier. They do not require a constructor, meaning the method can be called directly from the class. Consider the class below called `StringFun` that contains a method called `shout()` that returns the string provided in the parameters in all caps.

```
public class StringFun{
    public static String shout(String input){
        String temp = input.toUpperCase();
        return temp;
    }
}
```

The `shout()` function can be called directly from within the main method by referring to the class name. Below is an example of calling `shout()` from main.

```
public static void main(String[] args) {
    String result = StringFun.shout("Good morning, James");
    System.out.println(result);
}
```

information

missing semicolon

In the Hello World example, you have the following line:

```
System.out.println("Hello World!");
```

Try removing the semicolon at the end and see what happens. The code should now look like this:

```
System.out.println("Hello World!")
```

What does Eclipse show you? It places a red "X" on the line where something is wrong. The end of the line has a red underline. If you hover over it, Eclipse will tell you what the error is:



The screenshot shows the Eclipse Java Development Environment (JDT) interface. A Java file named `*HelloWorld.java` is open in the editor. The code contains a main method with a println statement. The line `System.out.println("Hello World!")` is highlighted in blue, indicating it is selected. A red 'X' icon is positioned next to the closing brace of the main method on line 8, and a red underline is under the closing brace of the println statement on line 7. A tooltip window is displayed, showing the error message: `✗ Syntax error, insert ";" to complete BlockStatements`. The status bar at the bottom of the IDE shows the path `File / HelloWorld.java`.

```
1 package com.jamesgosling.helloworld;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("Hello World!")
8     }
9
10 }
```

naming_variable

```
int players;  
String name;  
boolean enrolled;
```

Naming Variables



There are a few things to know when you want to create a name for your variable.

- **Meaning:** When creating a name for a variable, it is important to give the variable a meaningful name. For instance, if you wanted to store the number of days in a week, you might call the variable `daysOfTheWeek` rather than `thing`. This makes it easier for you to identify what the variable is.
- **Characters:** There are rules for naming variables:
 - Variable names must begin with a letter, a dollar sign `$`, or an underscore character `_`.
 - Subsequent characters may be letters, digits, dollar signs, or underscore characters.
- **cAsE sEnSiTiVe:** Variable names are case sensitive. For instance, `daysOfTheWeek` and `DAYSOFTHEWEEK` are considered two different variables.
- **Convention:** You may have noticed the strange capitalization for the variable names (e.g. `daysOfTheWeek`). This style of naming variables is known as *camel case*. When naming variables with compound words, the first word is all lowercase, and the first letters of each subsequent word are capitalized. While not required, most Java programs are written using this convention.

inheritance

Classes can use member variables and methods from another class via inheritance. For instance, a car and a truck are different in many ways, but they also share a few commonalities. They are both vehicles with a make, color, fuel level, and engine. Below are two simple classes named `Car` and `Truck` that have a member variable for make, color, fuel level, and engine state. The `Truck` class will have an additional member variable for the length of the flatbed. The `Car` class will have an additional member variable for the width of the trunk. Below is the final product of creating the two classes.

example

Car class:

```
public class Car {  
    String make;  
    String color;  
    boolean isRunning;  
    int fuelLevel;  
    int trunkWidth;  
  
    public Car(String make, String color, int fuelLevel, int trunkWidth) {  
        this.make = make;  
        this.color = color;  
        this.isRunning = false;  
        this.fuelLevel = fuelLevel;  
        this.trunkWidth = trunkWidth;  
    }  
}
```

Truck class:

```
public class Truck {  
    String make;  
    String color;  
    boolean isRunning;  
    int fuelLevel;  
    int flatbedLength;  
  
    public Truck(String make, String color, int fuelLevel, int flatbedLength) {  
        this.make = make;  
        this.color = color;  
        this.isRunning = false;  
        this.fuelLevel = fuelLevel;  
        this.flatbedLength = flatbedLength;  
    }  
}
```

```
public class Vehicle {  
    String make;  
    String color;  
    boolean isRunning;  
    int fuelLevel;  
  
    public Vehicle(String make, String color, int fuelLevel) {  
        this.make = make;  
        this.color = color;  
        this.isRunning = false;  
        this.fuelLevel = fuelLevel;  
    }  
  
    public void printDetails() {  
        System.out.println("The " + this.color + " " + this.make + " has a fuel level of " + this.fuelLevel + ". Is it running? " + this.isRunning);  
    }  
}
```

nested classes

Nested classes, or subclasses, are classes within other classes. This is useful when the developer wishes to logically group classes only where required, thereby increasing encapsulation. This also improved the readability of the code, making it easier to maintain when the application scales. A good example of when to create a nested class would be a Toolkit class. This class could contain sub-classes that organize the tools by sub-category. Perhaps the purpose of the Toolkit is to provide some string functionality as well as some math functionality. Let the nested classes be static as this will be a utility class which needs no instance. Below you'll find an example.

example

Now add a bit of functionality to this class to demonstrate it's usefulness. In the `StringAssist` nested class, create a string method called `whisper()` that takes in a string and uses the `toLowerCase()` method to return the string in lowercase. Create a similar method called `yell()` that returns the string in uppercase. In the `MathAssist` class, create a boolean method called `isEven()` that returns true if the number is even and returns false otherwise. Additionally, create a similar method called `isOdd()` that returns true if the number is odd, and returns false otherwise.

```
//outer class
public class Toolkit {
    // nested class (inner class)
    public static class StringAssist {

        public static String whisper(String line) {
            String temporaryString = line.toLowerCase();
            return temporaryString;
        }

        public static String yell(String line) {
            String temporaryString = line.toUpperCase();
            return temporaryString;
        }
    }
}
```

```
// another nested class (inner class)
public static class MathAssist {

    public static boolean isEven(int number) {
        int result = number % 2;
        // if the number has a remainder of 0, it's even
        if(result == 0) {
            return true;
        }
        // otherwise, it's odd
        else{
            return false;
        }
    }

    public static boolean isOdd(int number) {
        int result = number % 2;
        // if the number has a remainder of 0, it's even
        if(result == 0) {
            return false;
        }
        // otherwise, it's odd
        else {
            return true;
        }
    }
}
```

These methods can be invoked in the main class by using the dot notation similar to `System.out.println()`.

```
public static void main(String[] args) {
    String sampleString = "Hey There, Neighbor";
    int number = 9;

    // whisper
    String whisper = Toolkit.StringAssist.whisper(sampleString);
    System.out.println(whisper);

    // yell
    String yell = Toolkit.StringAssist.yell(sampleString);
    System.out.println(yell);

    // is even
    boolean isEven = Toolkit.MathAssist.isEven(number);
    System.out.println(number + " is even? " + isEven);

    // is odd
    boolean isOdd = Toolkit.MathAssist.isOdd(number);
    System.out.println(number + " is odd? " + isOdd);
}
```

override_final

The `@Override` annotation is a tool that can be used to override, or update, the methods that are inherited from the parent class. It aids in creating sensible defaults in the code. In the `Vehicle` class created above, there is a method called `printDetails()` that prints the information about the generic vehicle. Since this class is being inherited, it will print the information of the `Car` and `Truck` objects that inherit from it. However, the method fails to provide the specificity that is available in the `Car` and `Truck` classes, such as the trunk width and the flatbed length. Use the `@Override` annotation to change the `printDetails()` method in both classes to print the details of the specific vehicle that are inherited from the `Vehicle` class. For instance, the `Car` class will additionally print the width of the trunk when the `printDetails()` method is called.

exam

Car class:

```
public class Car extends Vehicle {  
    int trunkWidth;  
  
    public Car(String make, String color, int fuelLevel, int trunkWidth) {  
        super(make, color, fuelLevel);  
        this.trunkWidth = trunkWidth;  
    }  
  
    @Override  
    public void printDetails() {  
        System.out.println("The " + this.color + " " + this.make + " has a trunk width of " + this.trunkWidth + " and has a fuel level of " + this.fuelLevel + ". Is it running? " + this.isRunning);  
    }  
}
```

```
public class Truck extends Vehicle {  
    int flatbedLength;  
  
    public Truck(String make, String color, int fuelLevel, int flatbedLength) {  
        super(make, color, fuelLevel);  
        this.flatbedLength = flatbedLength;  
    }  
  
    @Override  
    public void printDetails() {  
        System.out.println("The " + this.color + " " + this.make + " has a flatbed length of " + this.flatbedLength + " and a fuel level of " + this.fuelLevel + ". Is it running? " + this.isRunning);  
    }  
}
```

```
public static void main(String[] args) {  
    Vehicle vehicle = new Vehicle("AcmeVehicle", "Gray", 5);  
    vehicle.printDetails();  
  
    Car car = new Car("AcmeCar", "Black", 15, 10);  
    car.printDetails();  
  
    Truck truck = new Truck("AcmeTruck", "White", 25, 20);  
    truck.printDetails();  
}
```

interface

```
public class Square implements Shape {  
    float length;  
  
    public Square(float length) {  
        this.length = length;  
    }  
  
    public float area() {  
        return length * length;  
    }  
  
    public String shapeName() {  
        return "Regular Quadrilateral";  
    }  
}
```

```
public class Circle implements Shape {  
    float radius;  
  
    public Circle(float radius) {  
        this.radius = radius;  
    }  
  
    public float area() {  
        return 3.14f * radius * radius;  
    }  
}
```

call the method

```
Shape newCircle = new Circle(4.5);  
System.out.println(newCircle.shapeName());  
Shape newSquare = new Square(4.5);  
System.out.println(newSquare.shapeName());
```

default methods

```
public interface Shape {  
    float area();  
    String shapeName() {  
        return this.getClass().getSimpleName();  
    }  
}
```

example

```
import java.lang.*;
import java.io.*;

class Deck implements Drawable {
    List<String> cards;

    public Deck(List<String> cards) {
        this.cards = cards;
    }

    public String getTopCard() {
        return cards.iterator().next();
    }
}
```

expl

```
import java.lang.*;
import java.io.*;

class Rectangle implements Area{
    double height;
    double width;

    public Rectangle(double height, double width){
        this.height = height;
        this.width = width;
    }

    public double getArea(){
        return height * width;
    }
}
```

try_resources

In an earlier topic you learned how to use the Try/Catch/Finally block to manage exceptions, but there is another use of `Try` operator. Sometimes an instance of a class needs to use some sort of external resource like a database connection or file on the hard drive. You don't want to leave those connections open after you are done with your communication to those resources so a standard practice was implemented. In Java 7 the interface `AutoClosable` was introduced that forces any class that implements the interface to include a `.close()` method. You previously learned about this same `.close()` method when discussing the `FileReader` class. It was called in order to close the connection to the file. When a class instance is called with the `Try` statement, the `.close()` method on that class will be called automatically. This makes resource management much simpler than remembering to close the connections yourself. Take a look at the previous example and how it could be rewritten to use Try with Resources:

example

```
public class HelloWorld {  
    public static void main(String[] args) {  
        String fileContentsRead = "";  
  
        try(FileWriter writer = new FileWriter("story.txt")) {  
            writer.write("a");  
        }  
        catch(Exception e){  
            e.printStackTrace();  
        }  
        try(BufferedReader reader = new BufferedReader(new FileReader  
("story.txt"))){  
            String line;  
            while ((line = reader.readLine()) != null) {  
                fileContentsRead = fileContentsRead + line;  
            }  
        }  
        catch(Exception e){  
            e.printStackTrace();  
        }  
        finally {  
            System.out.println(fileContentsRead);  
        }  
    }  
}
```



Factory

This commonly used pattern is categorized as a creational pattern. Thus, the purpose of this pattern is to create new objects. It's no coincidence that this creation tool is called a factory: it produces objects upon request.

1. Implement the `Animal` interface which contains a single void method called `speak()` which does not take in any parameters.

```
public interface Animal {  
    void speak();  
}
```

2. Next, create a few classes and implement the interface. In the code below, the `Cow`, `Dog`, and `Cat` classes are created and implement the `Animal` interface.

Cow class:

```
public class Cow implements Animal {  
    public void speak() {  
        System.out.println("Moo");  
    }  
}
```

Dog class:

```
public class Dog implements Animal {  
    public void speak() {  
        System.out.println("Bark");  
    }  
}
```

Cat class:

```
public class Cat implements Animal {  
    public void speak() {  
        System.out.println("Meow");  
    }  
}
```

3. Create a new factory to manage the creation of new objects. The factory below is named `AnimalFactory`. The factory needs to take a command which will determine which object is requested. For the code below, a string is checked which will determine which object is returned.

```
public class AnimalFactory {  
    public Animal getAnimal(String animalType) {  
        if(animalType.equalsIgnoreCase("cow")) {  
            return new Cow();  
        }  
        else if(animalType.equalsIgnoreCase("dog")) {  
            return new Dog();  
        }  
        else if(animalType.equalsIgnoreCase("cat")) {  
            return new Cat();  
        }  
        else {  
            // factory cannot create an unrecognized object  
            System.out.println(animalType + " is not recognized by Ani  
malFactory.");  
            return null;  
        }  
    }  
}
```

4. In the main method, create a new `AnimalFactory` factory object. Then, create new animals and call the `speak()` method on it.

```
public static void main(String[] args) {  
    // animal factory  
    AnimalFactory animalFactory = new AnimalFactory();  
  
    // produce a cow and speak  
    Animal cow = animalFactory.getAnimal("Cow");  
    cow.speak();  
  
    // produce a dog and speak  
    Animal dog = animalFactory.getAnimal("Dog");  
    dog.speak();  
  
    // produce a cat and speak  
    Animal cat = animalFactory.getAnimal("Cat");  
    cat.speak();  
}
```

Builder

A builder pattern is categorized as a *creational pattern*. It operates by creating complex objects in a sequence of smaller steps. Less-overloaded constructors are needed as a result. It also solves the problem of having a complex factory. If a factory were to produce objects that are complex, it would need to have a multitude of different states of the same object to be returned. Additionally, builders solve the problem of optional parameters, where method overloading can be reduced.

Take the `Employee` class below which contains nine member variables and a default constructor.

```
public class Employee {  
    private String firstName;  
    private String lastName;  
    private String phoneNumber;  
    private String streetAddress;  
    private String city;  
    private String state;  
    private int zipCode;  
    private String jobTitle;  
    private float salary;  
  
    public Employee(String firstName, String lastName, String phoneNum  
ber, String streetAddress, String city, String state, int zipCode, Str  
ing jobTitle, float salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.phoneNumber = phoneNumber;  
        this.streetAddress = streetAddress;  
        this.city = city;  
        this.state = state;  
        this.zipCode = zipCode;  
        this.jobTitle = jobTitle;  
        this.salary = salary;  
    }  
}
```

Next, the class illustrates the mandatory and optional parameters separately. The mandatory parameters are set in the default constructor while the optional parameters are set using setter methods. Getter methods exist for all properties in the class.

```
public class Employee {  
    // mandatory parameters  
    private String firstName;  
    private String lastName;  
    private String jobTitle;  
    private float salary;  
  
    // optional parameters  
    private String streetAddress;  
    private String city;  
    private String state;  
    private int zipCode;  
    private String phoneNumber;  
  
    public Employee(String firstName, String lastName, String jobTitle,  
        float salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.jobTitle = jobTitle;  
        this.salary = salary;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
}
```

```
public String getLastName() {
    return lastName;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public String getStreetAddress() {
    return streetAddress;
}

public String getCity() {
    return city;
}

public String getState() {
    return state;
}

public int getZipCode() {
    return zipCode;
}

public String getJobTitle() {
    return jobTitle;
}

public float getSalary() {
    return salary;
}
```

Since the builder is responsible for constructing the object, the constructor for the `Employee` class should be `private`. This puts the builder in full control of the object creation process. The builder class can be nested within another class, which is what will be done for the `Employee` class. The naming convention for a builder class is to append "Builder" to the class name (i.e. `EmployeeBuilder`). The `EmployeeBuilder` class will contain its own private copy of the member variables from the `Employee` class. The `EmployeeBuilder` object can replace the mandatory parameters in the `Employee` default constructor, which will set all member values from the `EmployeeBuilder` into the `Employee` class. All optional member variable methods can then be written inside of the new `EmployeeBuilder` class.

Updated default constructor:

```
// an EmployeeBuilder object passed into the default constructor.  
private Employee(EmployeeBuilder builder) {  
    this.firstName = builder.firstName;  
    this.lastName = builder.lastName;  
    this.jobTitle = builder.jobTitle;  
    this.salary = builder.salary;  
    this.streetAddress = builder.streetAddress;  
    this.city = builder.city;  
    this.state = builder.state;  
    this.zipCode = builder.zipCode;  
    this.phoneNumber = builder.phoneNumber;  
}
```

Nested class:

```
// EmployeeBuilder class, nested within Employee class
public static class EmployeeBuilder {
    // mandatory
    private String firstName;
    private String lastName;
    private String jobTitle;
    private float salary;

    // optional
    private String streetAddress;
    private String city;
    private String state;
    private int zipCode;
    private String phoneNumber;

    public EmployeeBuilder(String firstName, String lastName, String jobTitle, float salary) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.jobTitle = jobTitle;
        this.salary = salary;
    }
}
```

```

public EmployeeBuilder setStreetAddress(String streetAddress) {
    this.streetAddress = streetAddress;
    return this;
}

public EmployeeBuilder setCity(String city) {
    this.city = city;
    return this;
}

public EmployeeBuilder setState(String state) {
    this.state = state;
    return this;
}

public EmployeeBuilder setZip(int zipCode) {
    this.zipCode = zipCode;
    return this;
}

public EmployeeBuilder setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
    return this;
}
}

```

The public methods within the `EmployeeBuilder` class can be called outside of the `Employee` object. The final step for the `EmployeeBuilder` class is to create the object that has been completed. This method will be called `build()`. A method is created which will return the newly built `Employee` object.

Build method in EmployeeBuilder:

```

// last method in `EmployeeBuilder` class
public Employee build() {
    return new Employee(this);
}

```

Finally, the class will be properly tested by creating an Employee object in the main method using the `EmployeeBuilder` class to facilitate object creation.

```
public static void main(String[] args) {  
    // simple Employee object using only mandatory parameters  
    Employee johnEmployee = new Employee.EmployeeBuilder("John", "Smith", "Writer", 50000f).build();  
    System.out.println(johnEmployee.getFirstName() + " " + johnEmployee.getLastName() + " makes " + johnEmployee.getSalary() + " as a " + johnEmployee.getJobTitle());  
  
    // complex Employee object using additional setters from EmployeeBuilder class  
    Employee patEmployee = new Employee.EmployeeBuilder("Pat", "Green", "Chemist", 70000f).setPhoneNumber("555-123-4567").setStreetAddress("123 Alphabet St.").setCity("Tempe").setState("Arizona").setZip(85281).build();  
    System.out.println(patEmployee.getFirstName() + " " + patEmployee.getLastName() + " makes " + patEmployee.getSalary() + " as a " + patEmployee.getJobTitle() + " and lives in " + patEmployee.getCity() + ", " + patEmployee.getState());  
}
```

The first employee, `johnEmployee` is being created using the minimum number of variables. However, the remaining variables remain uninitialized. This can be a problem if the variables are null and undeclared. Below is an updated `EmployeeBuilder` constructor which sets a value to all variables.

```
// EmployeeBuilder class, nested within Employee class
public static class EmployeeBuilder {
    // mandatory
    private String firstName;
    private String lastName;
    private String jobTitle;
    private float salary;

    // optional variables
    private String streetAddress;
    private String city;
    private String state;
    private int zipCode;
    private String phoneNumber;

    public EmployeeBuilder(String firstName, String lastName, String jobTitle, float salary) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.jobTitle = jobTitle;
        this.salary = salary;
    }
}
```

```
//optional variables
this.streetAddress = "";
this.city = "";
this.state = "";
this.zipCode = 00000;
this.phoneNumber = "0000000000";
}

public EmployeeBuilder setStreetAddress(String streetAddress) {
    this.streetAddress = streetAddress;
    return this;
}

public EmployeeBuilder setCity(String city) {
    this.city = city;
    return this;
}

public EmployeeBuilder setState(String state) {
    this.state = state;
    return this;
}

public EmployeeBuilder setZip(int zipCode) {
    this.zipCode = zipCode;
    return this;
}

public EmployeeBuilder setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
    return this;
}

public Employee build() {
    return new Employee(this);
}
```

Visitor

The Visitor pattern is a type of *behavioral pattern* in which the visitor class will change the execution of code within the element class. The visitor class will pass in its own object into the element classes that implement the visitor class which will change the execution of code. The example for demonstrating the visitor pattern will model a weather alert system. Below is an interface which represents the `Alert` element.

```
public interface Alert {  
    public void alert(AlertVisitor alertVisitor);  
}
```

Next, concrete classes are needed to implement the `Alert` interface. The code below demonstrates `Smartphone`, `TV`, and `Radio` as the concrete classes. Note that the class `AlertVisitor` has not been created yet, but will momentarily.

Smartphone class:

```
public class Smartphone implements Alert {  
    public void alert(AlertVisitor alertVisitor) {  
        alertVisitor.visit(this);  
    }  
}
```

TV class:

```
public class TV implements Alert {  
    public void alert(AlertVisitor alertVisitor) {  
        alertVisitor.visit(this);  
    }  
}
```

Radio class:

```
public class Radio implements Alert {  
    public void alert(AlertVisitor alertVisitor) {  
        alertVisitor.visit(this);  
    }  
}
```

It's now time to create the `AlertVisitor` class. This is the visitor interface. The `AlertVisitor` will contain a method called `visit`, one for each class that is passed into the parameter. Since the `Smartphone`, `TV`, and `Radio` classes are implementing the visitor pattern, there will be three visitor methods containing a parameter for the three.

```
public interface AlertVisitor {  
    public void visit(Smartphone smartphone);  
    public void visit(TV tv);  
    public void visit(Radio radio);  
}
```

The final step is to create a concrete class for the visitor class.

```
public class AlertDisplayVisitor implements AlertVisitor {  
    public void visit(Smartphone smartphone) {  
        System.out.println("SMS: Weather conditions are rough in your area. Drive carefully.");  
    }  
  
    public void visit(TV tv) {  
        System.out.println("Channel 3 says: Stay indoors. Weather conditions are rough in the area.");  
    }  
  
    public void visit(Radio radio) {  
        System.out.println("WCME Radio says: stay indoors.");  
    }  
}
```

The visitor class can now be demonstrated in the main method by creating new alert concrete classes. Since they all implement the `alert` method, a new instance of the visitor class can be passed in (i.e. `new AlertDisplayVisitor()`).

```
public static void main(String[] args) {  
    // smartphone alert  
    Alert smartphoneAlert = new Smartphone();  
    smartphoneAlert.alert(new AlertDisplayVisitor());  
  
    // TV alert  
    Alert tvAlert = new TV();  
    tvAlert.alert(new AlertDisplayVisitor());  
  
    // radio alert  
    Alert radioAlert = new Radio();  
    radioAlert.alert(new AlertDisplayVisitor());  
}
```

Singleton Pattern

The singleton pattern is a common pattern you will see in Object Oriented languages like Java. The idea behind a singleton pattern is to have a class that can be initialized multiple times, but every instance is actually of the same object. Singletons are helpful for when you want all references within your code to point to the same resource. For example, say you had a database that you were connecting to. Creating a single connection to that database and running all queries through that connection would be ideal. Establishing a connection to the database each time that a query is about to be ran is a waste of resources and time.

```
public class DatabaseConnection {  
    private DatabaseConnection() {};  
  
    private static class DatabaseConnectionInnerClass {  
        private static final DatabaseConnection instance = new DatabaseConnection();  
    }  
  
    public static DatabaseConnection getInstance() {  
        return DatabaseConnectionInnerClass.instance;  
    }  
  
    // Code to connect and interact with the database goes here.  
}
```

There are many ways to construct a singleton, but this is the most common way due to it's "thread safe" nature. At minimum a Singleton class requires a private constructor instead of a public constructor and a static property that contains the instance of the object. The version of a singleton posted above uses an internal class to hold the instance of the static value which helps Java avoid concurrency issues. The instance of the class is not actually created until the `.getInstance()` function is invoked. This "wait until you're needed" approach is called "Lazy Loading".

To use a Singleton it's fairly straight forward. Call the `.getInstance()` method on the Singleton class name:

```
public static void main(String[] args){  
    DatabaseConnection db = DatabaseConnection.getInstance();  
  
    // Code that uses db variable goes here  
}
```

Strategy Pattern

The strategy pattern aims to give your code a way of deciding which behavior to execute when criteria is matched (similar to switch/case), but open for additional decisions to be added later. This is done through the use of an interface, concrete implementations of the interface, and flow control mechanisms such as for loop, while loop, and if statements.

Start with an interface called `PaymentStrategy` that defines a method called `.pay()` and returns void, as well as a `.getPaymentName()` method that returns a String:

```
public interface PaymentStrategy {  
    public void pay(double amount);  
    public String getPaymentName();  
}
```

Now add a couple of concrete payment classes that implement `PaymentStrategy`.

`CashStrategy.java` file:

```
public class CashStrategy implements PaymentStrategy {  
    @Override  
    public void pay(double amount) {  
        System.out.println(amount + " paid with cash.");  
    }  
  
    @Override  
    public String getPaymentName() {  
        return "Cash";  
    }  
}
```

CreditCardStrategy.java file:

```
public class CreditCardStrategy implements PaymentStrategy {  
    @Override  
    public void pay(double amount) {  
        System.out.println(amount + " paid with credit card.");  
    }  
  
    @Override  
    public String getPaymentName() {  
        return "Credit Card";  
    }  
}
```

Now to implement the Strategy Pattern, do the following in the main method of your application:

```
import java.util.*;  
  
public class Startup {  
  
    public static void main(String[] args) {  
        // Create a list of all possible payment strategies  
        List<PaymentStrategy> paymentStrategies = Arrays.asList(  
            new CashStrategy(),  
            new CreditCardStrategy()  
            // Add more strategies here later  
        );  
  
        // Use the console for user input  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Please enter a payment type.\n");  
        for (int i = 0; i < paymentStrategies.size(); i++) {  
            System.out.println("    " + paymentStrategies.get(i).getPaymentName());  
        }  
        String paymentType = scanner.nextLine().trim();  
        System.out.println("Please enter a payment amount.");  
        double paymentAmount = scanner.nextDouble();  
  
        // Using for loop instead of switch/case to match strategy  
        for (int i = 0; i < paymentStrategies.size(); i++) {  
            PaymentStrategy strategy = paymentStrategies.get(i);  
            if (strategy.getPaymentName().equalsIgnoreCase(paymentType)) {  
                strategy.pay(paymentAmount);  
            }  
        }  
        scanner.close();  
    }  
}
```

You start off by creating a list that holds all of the strategies. Then after collecting user input the application loops through all of the strategies until it finds a matching strategy. Once the matching strategy is found the `.pay()` method of the matching strategy is executed.

This code is much more extendable than a switch case. To add an additional payment option you only need to create another class that implements `PaymentStrategy` and add it to the list. The rest of the logic stays the same.

Template Pattern

The template pattern uses an abstract class to define methods that can be executed together as a group, then defines a single method to execute them. Then concrete classes are created that implement the abstract class, but only need to implement the individual methods that should be executed as one.

```
public abstract class Cooking {  
    abstract void mixIngredients(List<String> ingredients);  
    abstract void heat();  
    abstract void serve();  
  
    public final void cook(List<String> ingredients) {  
        mixIngredients(ingredients);  
        heat();  
        serve();  
    }  
}
```

Now to implement the abstract class consider the following:

```
public class BakeCookies extends Cooking {  
    @Override  
    void mixIngredients(List<String> ingredients) {  
        String ingredientNames = String.join(", ", ingredients);  
        System.out.println("Mixed: " + ingredientNames)  
    }  
  
    @Override  
    void heat() {  
        System.out.println("Baked at 375 degrees for 12 minutes.");  
    }  
  
    @Override  
    void serve() {  
        System.out.println("Served cookies on a plate.");  
    }  
}
```

Now finally to execute the template pattern method:

```
Cooking food = new BakeCookies();  
food.cook(  
    Arrays.asList(  
        "flour",  
        "baking soda",  
        "salt",  
        "butter",  
        "sugar",  
        "eggs",  
        "vanilla",  
        "chocolate"  
    )  
);
```

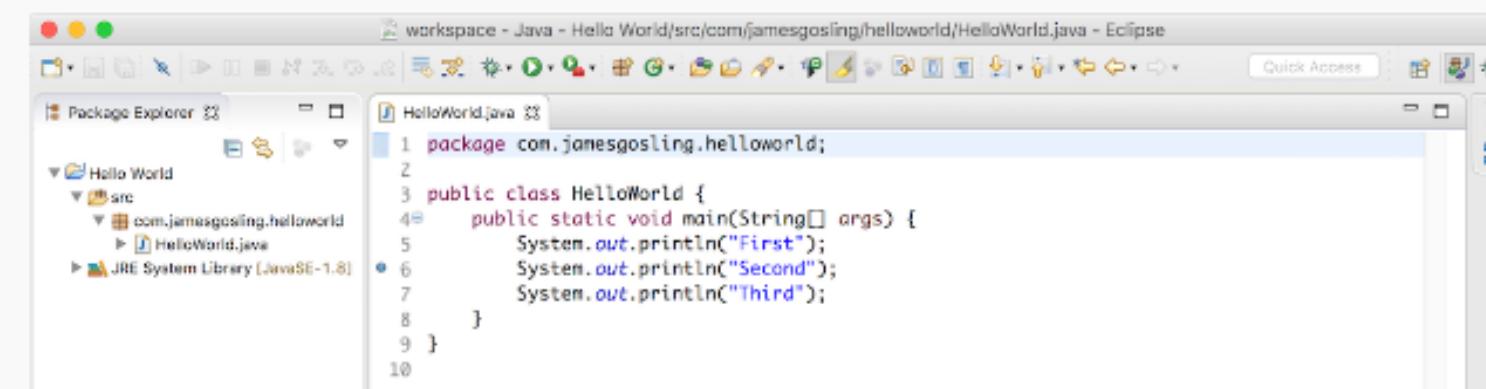
Debugging

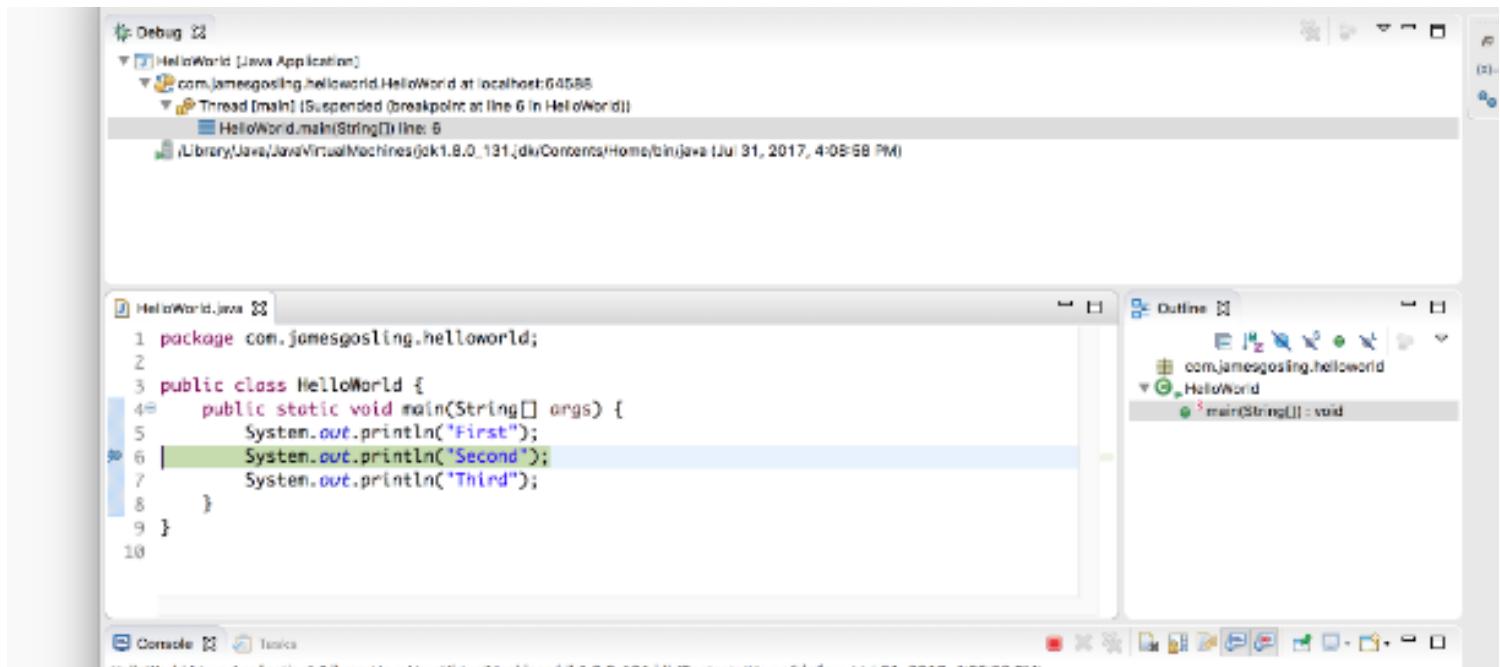
Application debugging is the act of stepping through an application one line at a time. This enables developers to locate the source of bugs and remove them (hence the name). Each line in a program can be paused during execution using a *breakpoint*. This is where the program execution will pause and allow for values and variables to be checked. A *breakpoint* can be added to a line in Eclipse by double-clicking the empty area to the left of the line number. This is where the application's state will pause *before* the code is run on that line.

Using the `print()` method for printing to the console is useful to display the values of variables, but proper debugging is quicker, more powerful, and scalable. Additionally, there are no guarantees that the `print()` method won't be the source of a bug.

```
public static void main(String[] args) {  
    System.out.println("First");  
    System.out.println("Middle");  
    System.out.println("Last");  
}
```

Now, add a breakpoint on the second print statement. Below is a screenshot of Eclipse with the code pasted and the breakpoint added on the second print statement.





Once the "Resume" button is clicked, the program will resume normal flow. If the code encounters a line with a breakpoint, the program will freeze as with the previous breakpoint. Replace the print statements in the main method with the code below. The code below is checking the value of a string and responding by running bits of code.

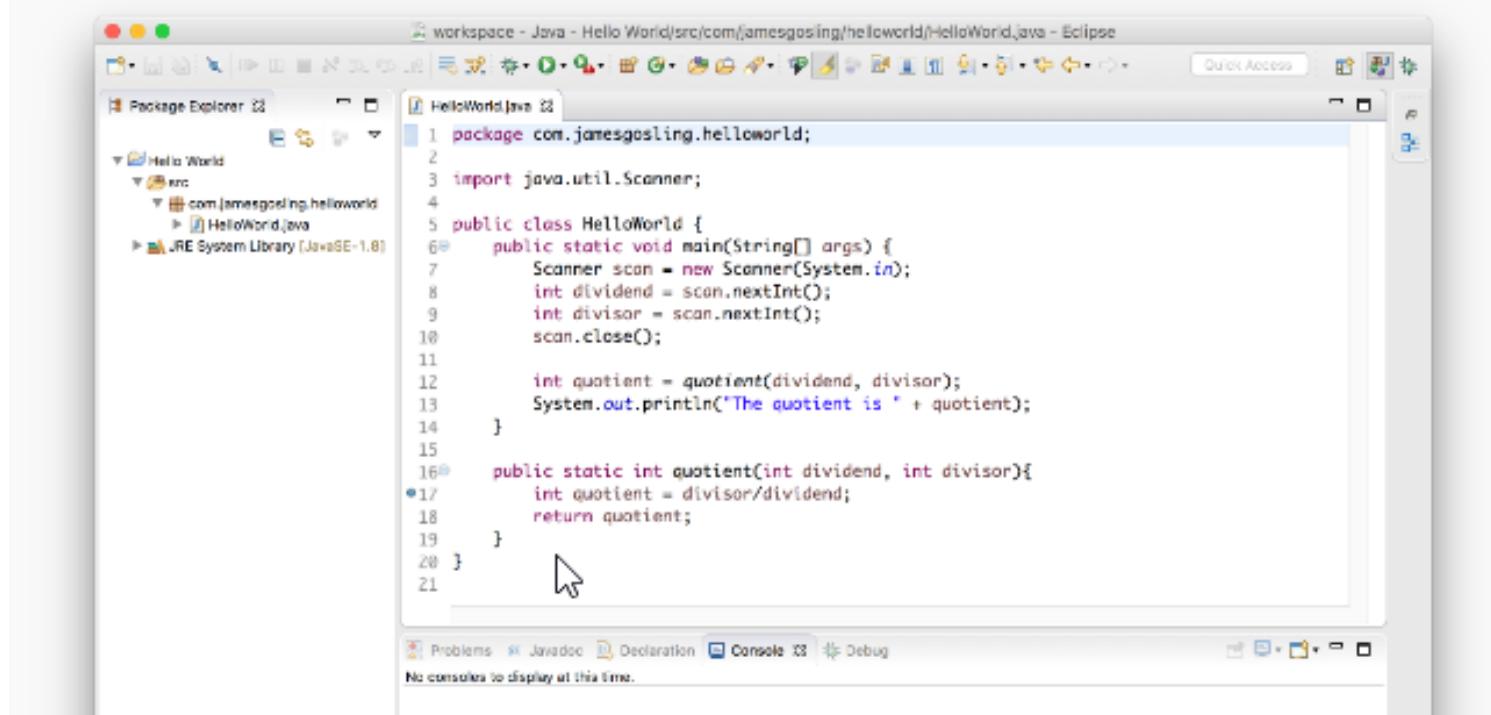
```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    int dividend = scan.nextInt();
    int divisor = scan.nextInt();
    scan.close();

    int quotient = quotient(dividend, divisor);
    System.out.println("The quotient is " + quotient);
}

public static int quotient(int dividend, int divisor){
    int quotient = divisor/dividend;
    return quotient;
}
```

The result should be 3; however, the value 0 is being returned: debugging to the rescue! This takes some investigative work to diagnose, but the issue stems from the `quotient` variable being printed. By working backward, the first point of failure occurs during the calculation of the `quotient` variable. Begin by adding a breakpoint where the `quotient` is being calculated.

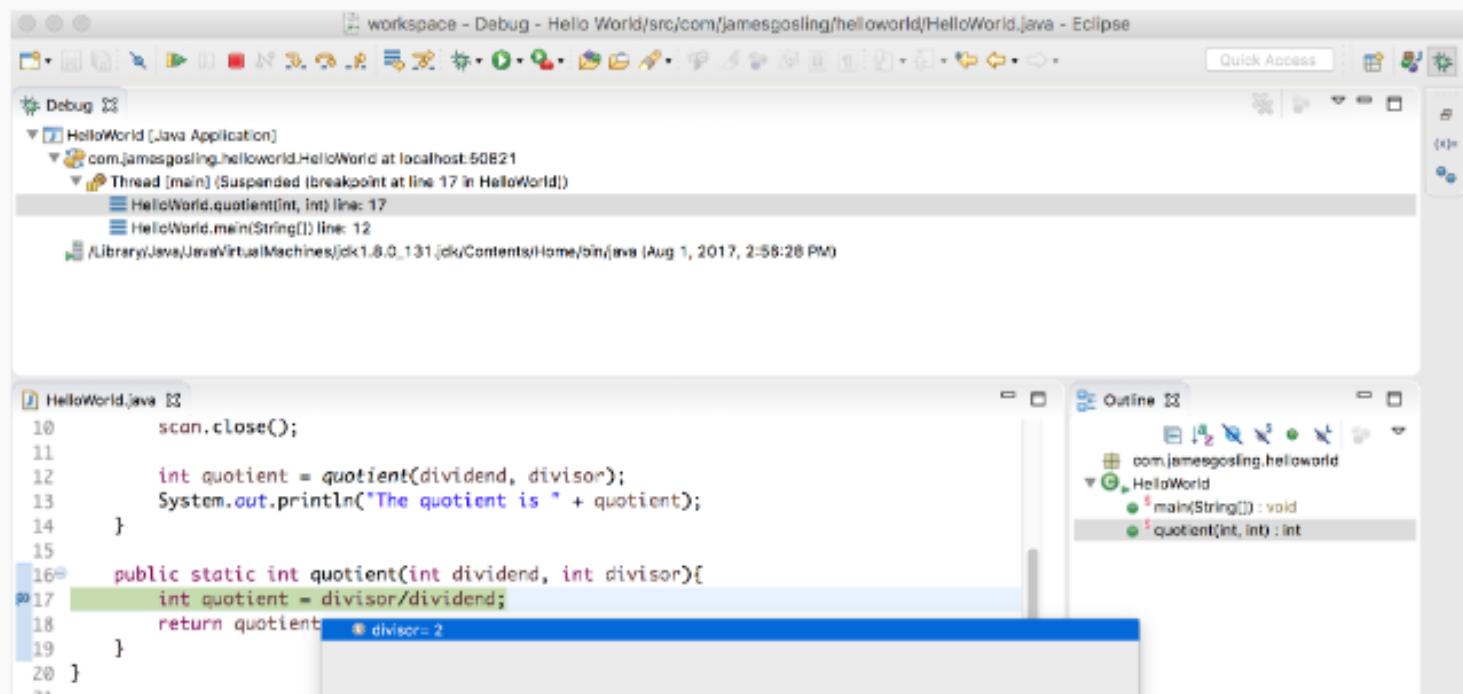
Debugging quotient:



The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** workspace - Java - Hello World/src/com/jamesgosling/helloworld/HelloWorld.java - Eclipse
- Package Explorer:** Shows a project named "Hello World" containing a package "com.jamesgosling.helloworld" which contains a file "HelloWorld.java".
- Editor:** Displays the Java code for "HelloWorld.java". A red dot (breakpoint) is visible on line 17, which contains the assignment statement `int quotient = divisor/dividend;`.
- Bottom Status Bar:** Shows tabs for Problems, Javadoc, Declaration, Console, and Debug. The Debug tab is selected, and the message "No consoles to display at this time." is shown.

This time, run the application in debugging and use the same input. The code execution should pause when the value of `quotient` is being set inside the `quotient` method. Now, hover the mouse over the `divisor` and `dividend` variables.



The value for the divisor is 2, and the value for the dividend is 6. The divisor is being divided by the dividend (i.e., 2/6). But, the dividend should be divided by the divisor (i.e., 6/2). The bug has been found. Now, the necessary updates need to be made without breaking the application. Fortunately, the code base is quite small here, and little testing is necessary. The updated `quotient` method should now look like the code below:

```
public static int quotient(int dividend, int divisor){
    int quotient = dividend/divisor;
    return quotient;
}
```

Rerun the application in debugging mode using the same inputs. Verify that the answer is correct and that the `dividend` and the `divisor` are correct. This section touches on the surface of debugging, but the mechanics are the same. As with the `quotient` variable, simply stepping backward through the program is an effective method of debugging an application to spot the area of the error.

Agile Project Management

As a technical professional, it's in your nature to continually pursue ways to improve, and the processes in which you create software are no exception. Project management has taken many different forms over the years, and the practice has never been as rapidly changing as it is now. Creating software products is often about exploring uncharted territory, accomplishing new achievements and creating new concepts. As such, developing software products is particularly hard to manage or forecast.

With software development being a collaborative practice, it is essential that you take an interest in project management and strive to find ways to improve your ability to predict and forecast deliveries. Keep in mind, however, that many people spend their entire career in project management, and so the scope of this course is to provide the background and a few common examples that you may encounter.

setup

<https://trello.com/>

Head on over to [Trello](#), to create an account.

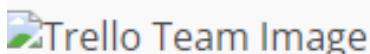
1. First, Click on the "Sign Up" button:



2. Then fill out the form:



3. You will then be prompted to create a team name. For now, choose any sample team name that comes to mind, you don't need to share it, and it can be changed later.



4. Once your team is set up, you will be asked to enter the emails of your team members, leave this blank and continue.



5. You will now be asked to choose your payment plan. You will use the free option as seen here. Then click continue.



6. Lastly, you will be able to confirm your registration. This is the final step so just hit continue.



The setup for this course is minimal. There are numerous project management tools available to you, but for simplicity sake, you will work within a web-based system called Trello. Trello is about as lightweight as they come, which is great for learning. There are plenty of more complex solutions that offer more features, but they are outside of the scope of this course.

Project Management and Agile

Project Management?

A project is temporary in that it has a defined beginning and end in time, and therefore defined scope and resources.

Therefore Project Management is defined as:

The application of knowledge, skills, tools, and techniques to produce activities to meet the project requirements.

Ok, that is a pretty broad definition, but if you think for a moment, you will realize that projects are ubiquitous in your life, regardless of how well *managed* they might be:

- Landscaping your yard
- Planning a party
- Remodeling your home

In these types of scenarios, you often have full control, so you'd give the project some thought and then dive in. However, if you were paying someone to perform these tasks, you would require that they are organized, efficient, timely and thorough. This is the task of project management.

Project managers are often tasked with the accountability for organizing a team and coordinating a project through to completion. Successful Project Managers are organized individuals who can foster the respect of the team and possess strength in communication and resourcefulness.

A key aspect of Project Management is **Risk Analysis**, in which the Project Manager strives to identify weaknesses in planning, or risks which may jeopardize the project timeline. When risks are identified, they should be discussed with the team and with stakeholders to determine the best course of action to *mitigate* the risk.

Therefore, while most are focused on leveraging skills to accomplish a specific task, the PM is focused on what could go *wrong* and ensuring that your team is prepared with an alternate plan.

A Program Manager is a title reserved for a Project Manager (PM), who oversees multiple large projects or divisions.

Responsibilities of project management

Scheduling

Scheduling and coordination of project resources are a core responsibility of Project Management to ensure that the project maintains its schedule. PMs will work with stakeholders and team members to coordinate and communicate the schedule, as well as adjustments to the schedule as they arise.

Scope Management

The *scope* of the project refers to the breadth of work to be done. Frequently, people underestimate or overestimate the scope of a project. This causes scope changes to be a frequent occurrence in the life of a PM. It is the PM's responsibility to manage and coordinate changes in scope to ensure that the outcome of the project is satisfactory.

Some scope adjustments are negotiable, such as a stakeholder requesting an additional feature. Others, however, are non-negotiable such as identifying a critical bug while working on a feature. The PM must negotiate and manage expectations within the team and stakeholders when scope changes occur.

Work Breakdown Structure

Stakeholders often deal in very high-level requests ("Let's build an email client"). The PM is required to break these high-level terms down into all of the sub-tasks which must be completed to satisfy the request.

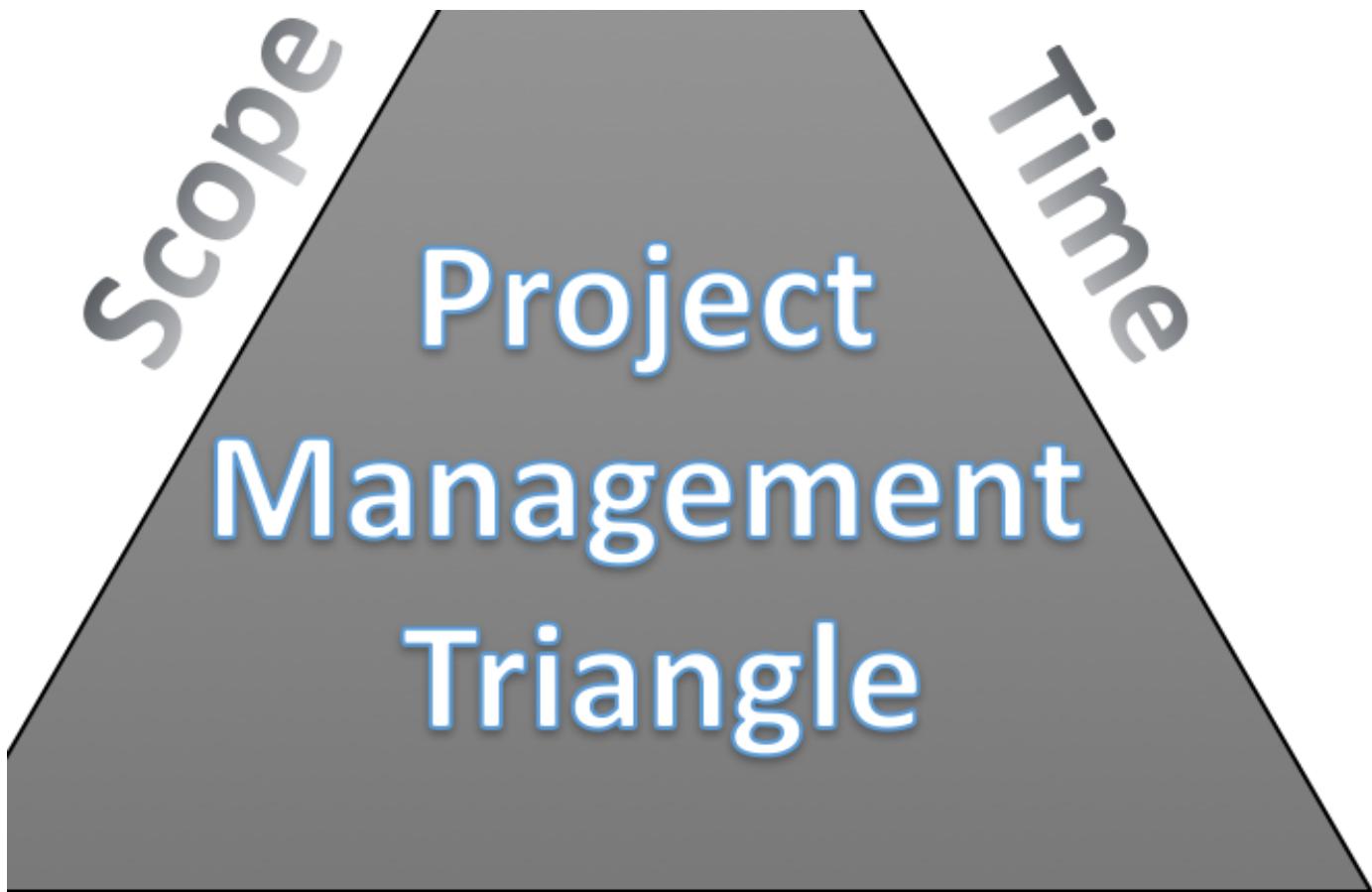
This process is called a Work Breakdown Structure (WBS). A WBS is the process of starting with the desired outcome and working your way back to define 100% of the dependent tasks to deliver that outcome. This process produces a comprehensive scope of the project.

A WBS is neither a project plan, a schedule, nor a chronological listing. It specifies what will be done, not how or when.

Project Management Triangle

Project Managers are accustomed to being asked to do more than is reasonable. Stakeholders want projects completed faster and cheaper, while the quality of the project is always expected to be paramount. This puts Project Managers regularly under pressure to deliver more with less.

The Project Management Triangle, also referred to as the Triple Constraint, or Iron Triangle is a diagram which emphasizes the give-and-take relationship between time, cost, and scope:



Cost

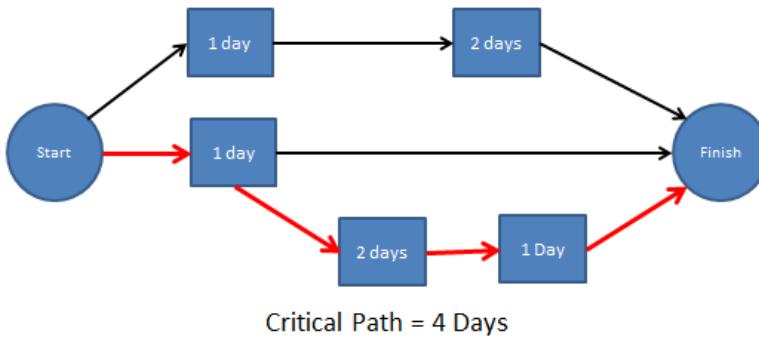


This illustration shows the relationship between the three primary constraints that affect the outcome of a project. If a stakeholder desires an earlier delivery, the triangle would suggest that either scope must be cut or the cost will increase, or both. Similarly, if the scope is increased, either the budget will need to increase, or the timeline must shift.

The Triple-Constraint helps remind you of the impact of your decisions and provides a common language to discuss the repercussions. However, the role of a Project manager is ultimately to find creative solutions, and therefore the triangle should serve as a guideline and not as a defense to avoid creative thought and critical problem-solving.

Critical Path

The Critical Path is a key element to managing and optimizing traditional Waterfall projects. The Critical Path is the sequence of tasks which conclude at the *end* of the timeline. This is the focus of the project manager to ensure you don't slip on this timeline. Consider the following Diagram of tasks and time to complete:



Notice that this is a 4-day project, wherein the top sequence of tasks complete in 3 days, yet the bottom (critical) path requires four days. Since the project is dependent on all tasks being completed, the entire project is on a 4-day timeline. The critical path is the sequence of tasks which limit the project timeline from being reduced. In other words, any extension of the tasks on the critical path will also extend the project deadline.

Twelve Principles of Agile Software Development

- . Your highest priority is to satisfy the customer through the early and continuous delivery of valuable software.
- . Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- . Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- . Business people and developers must work together daily throughout the project.
- . Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- . The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation.
- . Working software is the primary measure of progress.
- . Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- . Continuous attention to technical excellence and good design enhances agility.
- . Simplicity--the art of maximizing the amount of work not done--is essential.
- . The best architectures, requirements, and designs emerge from self-organizing teams.
- . At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Manifesto

The Agile manifesto has become a highly valued document in modern software development. Most current teams adhere to some form of Agile implementation, but the values stated in the manifesto are valued more often than not. Below, you will explore the 12 principles of the manifesto and get more familiar with the intent of the document.

Project Management Frameworks

Traditional / Waterfall

Traditional project management, also referred to as **Waterfall**, is the most natural to use. This is the way you probably visualize your landscaping project, as a sequence of tasks each dependent on the prior.

Lean

Lean Project Management finds its roots in Lean Manufacturing which is a process pioneered by Taiichi Ohno during his time as an industrial engineer with Toyota in the 1940s. The main principles behind Lean are to reduce waste and establish a continuous chain of production. This method is often referred to as **Just-in-Time** production.

Agile

Agile Project Management builds upon the foundation established by Lean and is viewed as its software-related counterpart. The main principle of **Agile Project Management** is incremental delivery and feedback from customers. It is an expansion on the Just-in-Time (JIT) principle of Lean. Incremental delivery principles can be traced back to the late 1950's. However, Agile gained tremendous favor with the software community in the early 2000s with the formalization of the Agile Manifesto: a document outlining the goals (not the processes) of Agile software development.

PMI

The **Project Management Institute (PMI)** is traditionally regarded as the primary source of knowledge and standards in regards to traditional Project Management. Their Project Management Body of Knowledge (PMBOK) is a book which outlines the recommended best practices.

PMI draws its project management knowledge from 10 areas:

- Integration
- Scope
- Time
- Cost
- Quality
- Procurement
- Human resources
- Communications
- Risk management
- Stakeholder management

That is to say that when managing a project, PMI asserts that these ten elements, and balancing them to a positive outcome are of primary concern.

PMI is an excellent source of knowledge; however, many Agile evangelists assert that the PMI's methods are too far rooted in traditional waterfall methods. This is a reputation that PMI is changing with more recent certifications such as PMI-ACP (Project Management Institute Certified Agile Practitioner).

Goals of Project Management

At this point you have undoubtedly realized the challenges of planning in software. It is challenging to predict and project all requirements and provide sufficient detail to ensure that the result meets expectations. Ultimately, every time you attempt up-front planning, you are inevitably wrong; you either overestimate or underestimate efforts. Project Managers have become conditioned to building *buffers* into their timelines to account for unforeseen risk. In the late 90's many companies began challenging the planning process and looking for a better solution.

Application Delivery Lag

In the mid-90's, industry experts asserted that the average delivery time of a non-trivial software product was **3 years**. So think back to your business requirements from the previous Hands-On. What if you didn't receive the resulting product for another three years? Think about the lost opportunities, the changing marketplace, how do you adapt to beat your competitors?

This delay between the identification of a product need, and the delivery of a software solution is referred to as Application Delivery lag. This was one of the primary drivers for software professionals to pursue a better way of managing their teams. How can you deliver value to the customer faster, and adapt more quickly to changing requirements?

The 80/20 Rule

When you perform significant, up-front planning to your projects, you'd often misjudge the customer's needs. The **80/20 rule** states that 80% of the value in your software comes from 20% of the effort. To explore this concept let's take an example: *Microsoft Excel*.

Microsoft Excel is an incredibly valuable product with many sophisticated features. If you wanted to create a competing product for Excel, it would take years to capture the decades worth of functionality that has been included in the system. Creating a 10-year timeline to recreate the functionality of Excel would probably not be a great strategy.

Instead, if you were to consider the most *valuable* functionality, you may come up with a top 3 list like this:

- User can enter values in cells
- User can perform basic math functions
- User can create pie charts from cell data.

These are some of the most commonly used and valuable features of Excel but certainly wouldn't take you years to complete. You could offer these features, as immediate value to customers within a few weeks.

Meanwhile, many of the features in Excel, which took years to develop, are rarely used. For example, the ability to write a function to determine if a cell contains a value *is* useful, but your users may prefer to not wait for a for that functionality before they get access to the top-3 above.

What if you built the most commonly used features first, and delivered those to your users before determining the next most valuable feature? This iterative approach is at the core of Agile Development.

The Birth of Agile

In 2001, a group of 16 software leaders attended a summit in Snowbird Utah to explore better ways of developing software. This group included Jon Kern, Kent Beck, Ward Cunningham, Arie van Bennekum, Alistair Cockburn, and eleven others, all well known today in the agile community.

This group of thought leaders set out to determine a process which will enable developers to get functionality into the hands of its users quickly and gather feedback. This way you would avoid the common pitfalls of significant, up-front planning or making incorrect assumptions of what the user wants.

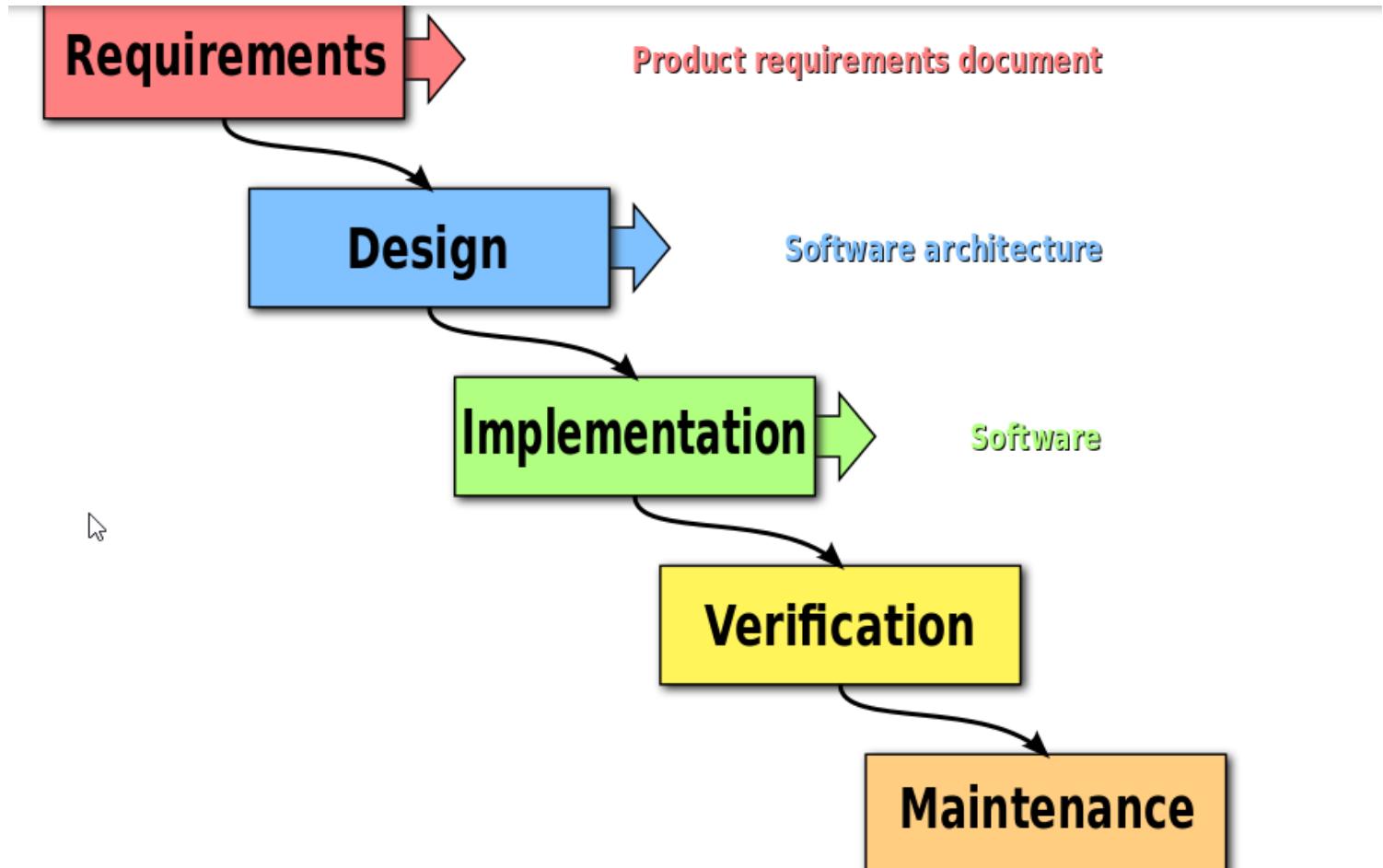
Much of their conversation was rooted in how to improve upon the traditional waterfall method and overcome its pitfalls. This now-famous encounter in Snowbird resulted in a document title the **Agile Manifesto**, which succinctly captures the principles set forth by the group. The manifesto is not an opinionated *process* but rather a statement of principles which enable development teams to welcome change with open arms instead of combating it and producing poor outcomes.

Waterfall

When you think about a project, you typically break it down into steps and sequentially perform those steps. This is the most natural and obvious approach to managing a project and tends to be familiar to most people. This non-iterative process of completing your work in phases is referred to as the Waterfall method due to the way the progress *flows* from one stage to the next. In this lesson, you will learn about the most common traditional method for managing projects.

The Waterfall methodology was initially defined by Winston W. Royce in 1970. The Waterfall Methodology is a natural approach to project management, and as such, it quickly became the norm as it was easy for all involved parties to comprehend. The premise of the waterfall method is that progress flows from stage to stage, each phase being dependent on completion of the previous stage. The specific implementation, or stages, of the process, differ significantly; however the notion of a sequential process is prevalent throughout.

This diagram illustrates a typical Waterfall approach to the Software Development Lifecycle.



Notice the waterfall-style flow of progress in this sequence of activities. Because there is no overlap (by design), each phase is dependent on thorough completion of the prior phase. The Waterfall method makes the assumption that all requirements can be gathered up front and thoroughly documented. This process is often well-received by stakeholders since it puts design in control of the process, and the business teams are forced to perform a comprehensive evaluation of the requirements up front. Once the requirements phase is complete, the control flows 'downhill'.

- The **Design** phase is dedicated to the in-depth documentation of architectural specifications, and user-interface designs as well as hardware specifications where applicable. The goal of this phase is to produce design documentation which can be turned over for implementation potentially by a separate team. Architects and designers negotiate the best approach to satisfying the business requirements and often cycle back and forth revising scope and approach.
- The **Implementation** phase is when all of the actual development takes place. This phase belongs to the programmers in the Waterfall method, as they take the project requirements and specifications, and code the applications.
- The **Verification** phase is dedicated to a series of quality assurance tests to validate that the application is working as designed and that all requirements have been met. This phase is often conducted by a separate QA team, who also has an interest in producing thorough documentation of the testing process and test cases.
- During the **Maintenance** phase, the customer is using the developed application. As problems are found due to improper requirements determination or other mistakes in the design process, or due to changes in the users' requirements, changes are made to the system during this phase.

Advantages to the Waterfall Model

Advantages of using the Waterfall method include:

- Technical documentation is thorough and complete, enabling newcomers to the team or acquiring businesses to have detailed training and reference guide.
- The process is a natural approach, requiring minimal training to explain the goals and expectations of the individual or team.
- This structured approach removes any doubt regarding accountability for each deliverable.
- Estimation of duration and cost are easy to calculate.

Disadvantages of the Waterfall Model

Disadvantages to using the Waterfall method include:

- Clients and stakeholders struggle to describe and consider every design requirement upfront.
- Changes to requirements most often have a direct negative impact on delivery, casting blame on whoever overlooked a detail.
- This process is often the longest to deliver the product as it requires so much planning.
- The process of redesigning, or re-engineering system features can become incredibly expensive.

Keynote

Risk Analysis	Strategy for predicting complications with your project plan.
Scope	The full requirements of the project.
Work Breakdown Structure	A description of the sub-tasks required for the project.
Project Management Triangle	The relationship between time, cost, and scope.
Waterfall	Traditional, phased project management.
Lean	A Just-In-Time management style emphasizing a reduction in waste.
Agile	A management style which favors iterative progress and welcomes customer feedback.
Manifesto	A declaration and description of beliefs and/or purpose.
Methodology	A process or approach to solving certain kinds of problems.

Four Pillars

Kanban is a method for managing the creation of products with an emphasis on regular delivery while not overburdening the development team. There are four simple principles to Kanban:

Visualize What You Do Today

Similar to Scrum's notion of transparency, Kanban enforces that the team has readily available reporting to provide visuals that describe the team's progress and bottlenecks.

Limit the Work-In-Progress

Instead of regularly-paced increments, Kanban employs a continuous delivery approach. When work is completed, and valuable functionality is customer-ready, why wait to deliver it? Limiting the work in progress helps the team to avoid over-committing on what can be delivered and quickly identifies bottlenecks.

Enhance Flow

To facilitate continuous delivery, the team's sole focus is on completing the work in progress, after which they will begin work on the next highest priority item in the backlog. Flow refers to the pace at which work flows through the team's workflow pipeline.

Continuous Improvement

Kanban promotes an emphasis on continuous improvement. By leveraging the visual nature of a Kanban system, you should strive to improve your efficiencies through a constant re-evaluation of your processes. This method should be adaptive and closely monitored for opportunities for the team to improve.

Workflow

In Software Development, Kanban is often adopted as an extension or a modification to Scrum processes. Kanban has a much less opinionated approach to software project management. Limiting WIP, continuous delivery and Flow are at the heart of Kanban. However, many other concepts such as the roles and ceremonies are open to the team's discretion. In this lesson, you will explore the specifics of a Kanban workflow.

Boards

The function of the Kanban board is to ensure the team's work is visual and available, their workflow is apparent, and all impediments are quickly identified. A basic Kanban Board will typically have three standard swim lanes:

To Do	Doing	Done
-------	-------	------

Many teams will expand on these three categories and add additional states. Notably, the *Doing* category is subject to being broken into multiple stages. As long as the workflow and policies are clearly stated, the team should modify the swim lanes to suit their needs.

Cards

In manufacturing, Kanban cards were implemented as a visual cue to the upstream vendor, indicating the needs of the downstream customer. These physical cards would document the specific needs and would be handed to the vendor. Software has adopted this format to suit your needs, and often Kanban Cards will resemble User Stories, with a familiar format:

The Kanban board will list all required work as Kanban Cards, explicitly stating the needs of the requester (customer). The cards will originate in the backlog, and move through the workflow based on priority and Flow.

The reason for using this card format is to ensure visibility to work being done. Recall that the first of your three Kanban pillars was to *visualize the work you do today*. Documenting that work and placing it on a visual board helps you to understand:

- What is being requested
- What stage of the workflow is it in
- Details of the request

Benefits

A few of the benefits of Kanban:

- Because it focuses on continuous improvement, it's likely that teams will continually yield high-quality results.
- Due to the focus on continuous improvement, both productivity and efficiency are likely to increase while reducing wasted time and resources.
- Goals can be more easily accomplished by Teams due to the visual nature of Kanban boards or cards.
- The decrease in waste leads to a higher-quality product or service at a cost-effective price for the consumer.

Key Terms

Kanban	A project management method with an emphasis on continual delivery and reducing waste.
Work-In-Progress (WIP)	The total of all work which is currently being done.
Flow	The throughput of work through the project management workflow.
Kanban Board	A visual representation of the Kanban project's current status.
Kanban Cards	Details and supporting documentation for work being requested.

Common Challenges

Estimation

The purpose of story pointing in a *Scrum* setting is to establish a velocity accurately. In a Kanban workflow, however, there is no concept of a sprint, therefore, no velocity. Kanban teams instead typically measure *Flow* or the *Flow Rate* as either the sum of story points in a given time-period or a ticket quantity.

Consider a scenario where you were to measure productivity as ticket volume. In this case, having equally-sized tickets would be essential to accurate metrics. For this reason, Kanban is often seen as a preferred methodology to teams which are tasked with maintaining an existing system. The measurement here is more historical than predictive.

Kanban can, at times, be more challenging to predict delivery of features.

Throttling Work in Progress

Without a Sprint structure, Kanban teams rely on limiting work in progress to throttle the flow of cards (stories). This can be a challenging concept to calibrate with a new team and can take time and revision to get to a proper setup.

The WIP limits help to protect the team by establishing manageable rates of throughput for tickets. In Scrum, the sprint planning meeting helps prevent overwhelming the team, as the team has an established velocity and a chance to voice concerns about the sprint backlog. In Kanban, the WIP limit is the only mechanism in place to curb product owner expectations and enable the team to have a healthy balance.

Unified Backlog

Sprint backlogs don't exist on a Kanban team since there is no concept of a sprint. Instead, the team can pull at any time, from the top of the product backlog. This requires the Product Owner to be highly involved in the day to day grooming of the product backlog. There is no opportunity for the Product owner to do a *big cleanup* on the backlog before a sprint planning.

The benefit of continuous flow is that the backlog can be much more flexible. The downside is that it requires much more effort by the product owner to keep it current.

Be Agile First, Be Kanban Second

Remember, your goal is to be an *Agile* software team. Kanban is a great workflow to achieve *Agile* principles. However, don't get too wrapped up in the process. Remember your ultimate goal is to be *Agile*, not to be *Kanban*.

Extreme Programming (XP)

You've now learned about two of the most common Agile frameworks in Scrum and Kanban. Extreme Programming (XP) is another common Agile framework centered around engineering principles and focused on ensuring delivery of high-quality software. XP teams work collaboratively in short development cycles and are flexible and adaptable to change. XP utilizes user stories and frequent small planned releases.

XP has twelve core practices grouped into four categories:

- **Fine Scale Feedback**
 - Test Driven Development
 - Planning Game (frequent release and iteration planning)
 - Whole Team (the customer is a member of the team and available at all times)
 - Pair Programming (two people should produce code)
- **Continuous Process**
 - Continuous Integration
 - Design Improvement (embrace refactoring)
 - Small Releases (deliver frequent releases of working functionality)
- **Shared Understanding**
 - Simple Design
 - System Metaphor (class and method should be named for functionality)
 - Collective Code Ownership
 - Coding Standards (entire team agrees on standards and holds one another accountable)
- **Programmer Welfare**
 - Sustainable Pace (people perform best when rested)

Project Planning

A project plan is your main tool for answering big-picture questions about your project. This is often the tool that is used to discuss the different needs of the departments in a company.

- This is a long-term plan that helps you answer some of the big questions your application will face
 - When should you start advertising?
 - When should you hire a QA tester?
 - Are there any bottlenecks that will cause your development to slow down?
 - If you hired another developer, how would that change the release date?
-

Gantt Charts

Being able to predict the future is very difficult, so it helps to break down the thing you are predicting into smaller parts. A Gantt chart is a way of seeing all your smaller pieces put together in a timeline so that you can see who is working on what and when.

Liquid Planner

The tool you will use to illustrate the idea of a project plan is [Liquid Planner](#). It is not a tool for tracking bugs or small tasks, but helps you see the big picture of your project. It is especially good for answering "what if" questions since it can recalculate your estimated completion date automatically and adjust your timelines as you adjust priorities. Not all Gantt chart systems have this capability, and for some scenarios, it is not even something you would want to happen.

Estimates

The upper case E indicates when a task is estimated to be complete. This is an estimate because for each task you can put in an optimistic and pessimistic time estimate. This is an important part of project management, things rarely go according to plan, and building in some space can make your plan much more resilient to change.

E

keyterm

Gantt Chart A way of visualizing tasks in relation to each other.

Critical Path The key tasks in your project that form the minimum time needed to complete the project. Delay on any of these tasks will definitely change your project's completion date.

Tracking time

Since Liquid planner will automatically adjust your tasks based on who is assigned and the dependencies you have set up, you must manually track the completion of the tasks you work on. Say DavidK works on the [Design course content](#) task for 10 hours. This would adjust the overall project to look like this.

Deadlines

Since the project time will naturally just push back when no time is logged, it is important to enter deadlines for different parts of your project. This will tell you when you are at risk of missing these dates and give you a chance to address it.

Now, return to the original scenario that is estimated to be completed on September 24th.

mysql

ds

aggregate functions subqueries

To **aggregate** is to gather or form something from a collection of other items.
A **function** is a tool that accepts input parameters, performs actions and returns a result.

Common Aggregate Functions

There are a variety of aggregate functions available in SQL. You will learn some of the most common ones:

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()

All of these functions are self-describing and share the same syntax: `name of the function (input parameter)`

```
SELECT protocol,
    SUM(traffic_in) AS traffic_in,
    SUM(traffic_out) AS traffic_out
FROM traffic
GROUP BY protocol
HAVING traffic_in > traffic_out
ORDER BY protocol
```

I

```
select account_holder, round(amount *0.05) as interest
from accounts
```

avg

```
SELECT AVG(grade)
FROM student
WHERE enrolled_course = 'programming_101';
```

Just like with SUM(), we did NOT use the * wildcard. You need to specify the column you want to obtain the average for as the input. AVG will only properly work with numeric columns.

```
SELECT AVG(input_parameter)
FROM table_name
WHERE condition;
```

```
SELECT AVG(input_parameter)
FROM table_name
WHERE condition;
```

The `avg()` function will return the average value of a numeric column. average:

```
SELECT avg(amount) FROM sakila.payment
```

Output:

A screenshot of a MySQL command-line interface. The top part shows the SQL query: "SELECT avg(amount) FROM sakila.payment". Below the query, there are zoom and refresh controls, and a status bar showing "100%" and "39:1". The bottom part shows the results in a grid. The first row is labeled "Result Grid" and includes icons for Result Grid, Refresh, Filter Rows, and Search. The data grid has two rows. The first row contains the column header "avg(amount)". The second row contains the value "4.200667". A navigation arrow is visible on the left side of the grid.

avg(amount)	
▶	4.200667

count

COUNT() returns the total number of records that matches the given criteria.

example

```
SELECT COUNT(*)
FROM student
WHERE enrolled_course = 'programming_101';
```

In this case, we use the a * as a wildcard. This is acceptable because it does not matter what column name you choose to count as long as the condition is met. (`enrolled_course = 'programming_101'`)

If there are 20 students enrolled in "programming_101", then count function would return 20 .

syntax

```
SELECT COUNT(input_parameter)
FROM table_name
WHERE condition;
```

The count() function will return the number of rows that match certain criteria.

```
SELECT count(amount) FROM sakila.payment  
WHERE amount < 4.00
```

And the output will be:

The screenshot shows a MySQL command-line interface. At the top, there are two lines of code:

```
1 SELECT count(amount) FROM sa  
2 WHERE amount < 4.00
```

Below the code, there is a progress bar indicating the query is running. On the left side of the interface, there are zoom and refresh controls, followed by the text "100%" and "20:2".

At the bottom, there is a toolbar with the following buttons and labels:

- Result Grid** (highlighted in yellow)
- Filter Rows:**
- Search**

The main area displays the results of the query:

count(amount)
8303

min

MIN() returns the smallest value a numeric column.

```
SELECT MIN(input_parameter)
FROM table_name
WHERE condition;
```

In the following example, you will see how to use the minimum function to return the lowest grade of students who are enrolled in a course called "programming_101". Assume that the table is called "student" and there are 5 columns (student_name, enrolled_course, tuition, grade, enrollment_date):

```
SELECT MIN(grade)
FROM student
WHERE enrolled_course = 'programming_101';
```

```
SELECT min(amount) FROM sakila.payment
```

And that would give the following output:

The screenshot shows a terminal window with a dark background. At the top, the command `SELECT min(amount) FROM sakila.payment` is entered. Below the command, the number '1' is displayed with a blue dot next to it, indicating the current row. The number '2' is also visible. In the bottom left corner, there are zoom controls (100%, 39:1) and a scroll bar. At the bottom of the window, there are buttons for 'Result Grid', 'Filter Rows:', 'Search', and 'Export' with icons. The results table has one row with the header 'min(amount)' and a single data cell containing '0.00'.

	min(amount)
▶	0.00

sum

SUM() returns the **total value** of the sum of a numeric column that matches the given criteria.

example

```
SELECT SUM(tuition)
FROM student
WHERE enrolled_course = 'programming_101';
```

In this case, we did NOT use the * wildcard. You need to specify the column you want to obtain the sum for as the input. Sum will only properly work with numeric columns.

sytx

```
SELECT SUM(input_parameter)
FROM table_name
WHERE condition;
```

The `sum()` function will return the total sum of a numeric column. all totals paid within the payments by running the following query

```
SELECT sum(amount) FROM sakila.payment
```

And the output will be:

A screenshot of a database query results interface. At the top, there is a code editor window containing the SQL query: `SELECT sum(amount) FRO`. Below the code editor are zoom and refresh controls, showing "100%" and "39:1". To the right of these controls are buttons for "Result Grid" (with a grid icon), "Filter Rows" (with a search icon), and a refresh icon. The main area displays a result grid with one row. The first column is labeled "sum(amount)" and contains the value "67416.51". There is also a play button icon to the left of the first column.

sum(amount)
67416.51

what the different between sum and count

Count is used to find the total NUMBER of a given input. Sum is used to find the total SUM of a given input. If you have a list of 5 numbers (1 through 5), the count of the numbers is 5 because there are 5 values. However, the sum of the numbers is 15. This is because that is the total of the values. ($1 + 2 + 3 + 4 + 5 = 15$)

max

MAX() returns the largest value a numeric column.

```
SELECT MAX(input_parameter)
FROM table_name
WHERE condition;
```

In the following example, you will see how to use the maximum function to return the highest grade of students who are enrolled in a course called "programming_101". Assume that the table is called "student" and there are 5 columns (student_name, enrolled_course, tuition, grade, enrollment_date):

```
SELECT MAX(grade)
FROM student
WHERE enrolled_course = 'programming_101';
```

```
SELECT max(amount) FROM sakila.payment
```

Which would give the output:

```
1 SELECT max(Total) FROM invoices;
```

Rows 1

max(Total)

25.86

Group BY_ Order by

The *Group By* statement is not technically an aggregate function. However, it is worth exploring because it is used very often in conjunction with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Now that you are familiar with the syntax, imagine how you would write the SQL query for the problem presented previously with a little more detail included, "Find the number of students enrolled in each course. Include the name of the course in the results." Give it a try and then see below for comparison (Hint: you will need to use `count()`):

```
SELECT COUNT(student_name), enrolled_course
FROM student
GROUP BY enrolled_course;
```

group_by

The SQL GROUP BY statement is often used with the functions mentioned earlier in this lesson (MIN, MAX, SUM, AVG, COUNT). The results of these functions can then be grouped using one or more columns. Below is the syntax:

```
SELECT column_name(s) FROM table_name  
WHERE condition  
GROUP BY column_name(s)
```

Using GROUP BY, run the following query to list the number of addresses in each district:

```
SELECT count(address_id), district FROM sakila.address  
GROUP BY district  
ORDER BY count(address_id) DESC
```

```
SELECT sakila.customer.customer_id,sakila.customer.first_name, sakila.customer.last_name, COUNT(rental_id) AS TotalRentalCount FROM sakila.rental  
INNER JOIN sakila.customer USING (customer_id)  
GROUP BY sakila.customer.customer_id
```

```
SELECT customer_id, sum(amount) AS TotalRentalAmount FROM sakila.payment  
GROUP BY customer_id
```

```
SELECT customer_id, sum(amount) AS Total, count(rental_id) AS NoOfRentals FROM sakila.payment  
GROUP BY customer_id  
HAVING count(rental_id) > 40
```

subqueries

A *subquery* is a SQL query that is nested in another query.

Why you should use it

Subqueries are used to solve a problem that would require another separate query to obtain data needed for the primary (or parent) query. Instead of writing separate queries, you could write a subquery that would provide input for the larger query.

usefull

<https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php>

Batch SQL Jobs

In real-world practice, there is often a need to run multiple SQL queries sequentially. An example may be that new data is aggregated from multiple databases to populate the data in a web application.

In these instances, you need *SQL jobs*. A SQL job is a specified series of operations that are performed sequentially for a purpose.

Batch SQL jobs are multiple SQL jobs run at the same time. With all of these SQL jobs running and performing SQL tasks, there is the risk that something could go wrong. These jobs must be **monitored** to ensure they are completed successfully. Batch SQL jobs can be monitored manually but running them and then waiting for them to complete. Depending on the complexity of the SQL jobs and the amount of data, these jobs can take hours to complete. Therefore, batch SQL jobs are usually scheduled. Usually, a database administrator or other database specialist will write SQL scripts whose purpose is to monitor these jobs and advise if there is a failure or error.

common_colum_type

CHAR(size)	Holds a fixed amount of characters in a string (can contain letters, numbers and special characters). The size is defined within the parenthesis and can store up to 255 characters. If the datatype is defined CHAR(5), the string HAS to hold 5 characters.
VARCHAR(size)	Holds a variable amount of characters in a string (letters, numbers, special characters). The maximum size is located within the parenthesis and can hold up to 255 characters. If the datatype is defined VARCHAR(20), the string can hold up to 20 characters. The Unicode for VARCHAR is one per character.
NVARCHAR(size)	Same as VARCHAR but the Unicode is two per character.
INTEGER	Defines that the data stored in that column has to be an integer.
DATETIME	Defines that the data stored in that column is in the date time format: YYYY-MM-DD HH:MM:SS
NUMERIC(n,n)	Defines how many numbers can live on either side of the decimal point in a number. If numeric is defined like NUMERIC(10,2), then there can be up to 10 numbers on the right and left side of the decimal point and up to 2 numbers on the right side of the decimal. The largest number allowed in this case would be 99,999,999.99

create a table

Great! Now you understand the basic structure of a table, you will build one! You are going to be using the `CREATE TABLE` keywords to create a table in your existing database. Below is the syntax:

```
CREATE TABLE table_name (
    column1_name datatype,
    column2_name datatype,
    column3_name datatype,
    column4_name datatype,
    ...
)
```

The column parameters will define the column names in the table. The datatype will specify the type of data each column can hold. Below is a list of common data types used:

- `CHAR(size)`
- `VARCHAR(size)`
- `NVARCHAR(size)`
- `INTEGER`
- `DATETIME`
- `NUMERIC`



constraints

Constraints are used to specify rules for data in a table that will limit the type of data that can go within a table or column. This will make sure that the data is reliable and accurate and if any of the constraints are violated, the action or query will be aborted. You can apply a constraint to a column or a table. Read on to explore commonly used constraints.

autoinrecment

AUTOINCREMENT will automatically generate a unique number when a new record is inserted into a table. Often, the **PRIMARY KEY** field will also have AUTOINCREMENT since the primary key is the unique id for each row. It is handy to be able to generate the unique id for each row automatically.

```
CREATE TABLE AppUsers4 (
    AppUserID INTEGER PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    SignUpDate DATETIME NOT NULL)
```

check

CHECK is an expression that you create that will check each entry into the table for validation. If the entry fails the constraint, then the update or insert will fail. An example here would be a check constraint that would validate a phone or zip has the right number of digits, and that they were in fact digits instead of text. Here is an example:

```
CREATE TABLE customerExample(
    FirstName NVARCHAR(40),
    LastName NVARCHAR(30),
    State NVARCHAR(10),
    PostalCode INTEGER,
    FOREIGN KEY(State) REFERENCES states(Abbreviation),
    CHECK (length(PostalCode) = 5)
);
```

default

`DEFAULT` defines default values that you can assign to a column, in the case where an insert statement does not include a value for that column. A common use for this scenario is an Inserted column that you might have in your table, to indicate when the record was written. You might set the default constraint to the `datetime()` function, which would populate the date and time (local time, not UTC) of when the record was created in the table. An example would look like this:

```
CREATE TABLE customerExample(
    FirstName NVARCHAR(40),
    LastName NVARCHAR(30),
    State NVARCHAR(10),
    PostalCode INTEGER,
    SignUpDate datetime DEFAULT current_timestamp,
    FOREIGN KEY(State) REFERENCES states(Abbreviation),
    CHECK (length(PostalCode) = 5)
);
```

droping_table

Now that you know how to create a table and insert data into it, you will learn how to delete the table. The syntax is relatively straightforward:

```
DROP TABLE table_name;
```

Try out this syntax by deleting the table `actorExample` you created earlier:

```
drop table actorExample
```

foreign key

FOREIGN KEY is a relation between tables that will restrict the insertion of data in one table, depending on the contents of a second table. The **FOREIGN KEY** refers to the **PRIMARY KEY** in a separate table.

Say you have a Customer Table that has a column for a state of residence. You might have a Foreign Key to a table with all the states and their abbreviations in it, to ensure that any entry into the Customer Table has a valid state entry. Consider the following tables:

example

FirstName	LastName	State
Alex	Smith	MA
Ryan	Williams	AL
Bob	Jones	RI

Below is how you would use `FOREIGN KEY` when you are creating the customers table:

```
CREATE TABLE customerExample(
    FirstName NVARCHAR(40),
    LastName NVARCHAR(30),
    State NVARCHAR(2),
    FOREIGN KEY(State) REFERENCES states(Abbreviation)
);
```

Alabama	AL
Alaska	AK
Arizona	AZ
Arkansas	AR
California	CA
Colorado	CO
Connecticut	CT

insert

Now that you understand how to create a table, you can use **INSERT**, which you learned previously in this module.

Run the below query:

```
create table actorExample(
    actor_id smallint(5) unsigned,
    first_name varchar(45),
    last_name varchar(45),
    last_update timestamp)
```

Now you can insert data into this table:

```
insert into actorExample
values (200, "Jamie", "Thomas", "2020-01-23 12:16:34")
```

And if you look at the data in the **actorExample** table, you should see the below output:

200	Jamie	Thomas	2020-01-23 12:16:34
-----	-------	--------	---------------------

all_columns

If you wanted to insert all columns from one table to another, you would do this in general terms:

```
INSERT INTO table2  
SELECT * FROM table1  
WHERE condition;
```

some_columns

If you wanted to insert only some columns, you would do this in general terms:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1;
```

You could also add a `where` statement to this:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

```
INSERT INTO actorExample (actor_id, first_name, last_name, last_update)
SELECT actor_id, first_name, last_name, last_update
FROM actor;
```

not_null

`NOT NULL` will force that column not to accept null values. By default, columns can hold `NULL` values. `NOT NULL` will prevent that column from having a null value. This is useful when you are gathering data from a user (like signing up for a website), and you need every input field to contain some data.

Below is how it will look in your query:

```
CREATE TABLE AppUsers1 (
    AppUserID INTEGER,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    SignUpDate DATETIME NOT NULL)
```

Above, you are adding the `NOT NULL` constraint to the `FirstName`, `LastName` and `SignUpDate` columns. This will ensure that each of those rows in the `AppUsers` table will have data.

primary key

PRIMARY KEY is a combination of **NOT NULL** and **UNIQUE**, so that every row in the table is unique and not null. This will enable you, when working in other tables, to identify that the data is related to the original table. For example, a customer table may have an id column that is a primary key. You could then use that customer id in a table that lists purchases, to relate a particular purchase to an individual customer account. **Almost all tables should have a primary key.** Without a primary key, there is no way to identify each row uniquely.

All in all, **Primary Key** is a constraint that defines a column in a row so that it is not null and unique for every individual row. The SQL syntax for Primary Key is shown below:

Consider below:

```
CREATE TABLE AppUsers3 (
    AppUserID INTEGER PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    SignUpDate DATETIME NOT NULL)
```

What you have done above is replace **UNIQUE** with **PRIMARY KEY**. This will now ensure that the **AppUserID** is unique to every row and also is required to have a value.

unique

Unique ensures that some column or combination of columns is unique to each row in the table. It's similar to a primary key, but a table may only have one Primary Key, whereas it might have several Unique Keys. An example here might be a column containing a social security number. The primary key might be id, but you might also want to monitor that the social security numbers that are being inputted are also unique to a particular customer. Since social security numbers SHOULD be unique, a failure to insert because of a violation of such a unique key could be a way to sound an alert.

Consider below:

```
CREATE TABLE AppUsers2 (
    AppUserID INTEGER UNIQUE,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    SignUpDate DATETIME NOT NULL)
```

views

A **View** is a virtual table based on the result of a SQL statement. It contains rows and columns, just like a real table, but when a view is created, a table is NOT created; you are just showing the output of the selected data. The fields in the View are from one or more real tables that exist in the database. You can use SQL functions, **WHERE**, and **JOIN** statements in a view to be able to present the data as if it were coming from one table.

create a dumbuser

Creating a View using a Join will look very similar to what you have done in the past with Joins. Below is a Join query you ran in previous lessons:

```
SELECT first_name, last_name, film_id  
FROM sakila.actor  
INNER JOIN sakila.film_actor  
ON sakila.actor.actor_id = sakila.film_actor.actor_id
```

All you would need to do if you wanted to create a view with the above query is add the [Create View](#) statement on the first line:

```
create view ActorFilms as  
SELECT first_name, last_name, film_id  
FROM sakila.actor  
INNER JOIN sakila.film_actor  
ON sakila.actor.actor_id = sakila.film_actor.actor_id
```

And if you run:

```
SELECT * FROM ActorFilms;
```

You will easily be able to see the three columns selected in the CREATE VIEW query.

drop_views

Now, if you didn't need your view anymore, you would use the following generalized syntax:

```
DROP VIEW [view_name];
```

Practice by dropping the `CurrentCustomers` view you created:

```
drop view CurrentCustomers
```

syntax

Here is the generalized syntax for creating views:

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

And now you can try actually creating a view:

```
create view CurrentCustomers as  
select customer_id, first_name, last_name  
from customer  
where active = 1
```

Above, you are creating a view to see all customers ID, first name and last name who have an ID less than 30. When the above query is executed, the green checkmark will appear, so you know the view was created correctly. Next, to see the view, run the following query:

```
SELECT * FROM CurrentCustomers;
```

You will now see the CustomerId, FirstName and LastName of customers that are active. Creating views are useful, because it means you don't need to run complicated queries many times. You can run a complicated query and set it to a view that you can then select easily.

example

```
CREATE TABLE AppUsers (
    AppUserID INTEGER,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    SignUpDate DATETIME
)
```

The above query will create a table with four columns, but with no data, as you can see below:

The screenshot shows a tree view of a database schema. At the top level is a database icon labeled 'sakila'. Below it is a 'Tables' folder containing three entries: 'actor', 'address', and 'appusers'. The 'appusers' entry is expanded, revealing its 'Columns' folder which contains four entries: 'AppUserID', 'FirstName', 'LastName', and 'SignUpDate'. Below the 'Columns' folder are three more collapsed folders: 'Indexes', 'Foreign Keys', and 'Triggers'. A vertical scroll bar is visible on the right side of the interface.

crud

CRUD name	SQL command
Create	Insert
Read	Select
Update	Update
Delete	Delete

delete

Delete is a way to remove existing data from a table. Below is the syntax for deleting:

```
DELETE FROM table_name  
WHERE condition;
```

Well, Jonathan (Johnny) is now no longer an actor, and you've been asked to delete him from the `actor` table. Below is the query to delete Johnny from the table:

```
delete from actor  
where actor_id = 201;
```

info

DML	Stands for Data Definition Language used to define data structures.
DDL	Stands for Data Definition Language used to manipulate the data itself.

insert

The `INSERT INTO` statement is used to insert data into a table within a database. There are two ways to insert data into the database.

The first way is to specify both the columns and the values of the columns:

```
INSERT INTO table_name (column1, column2, column3...)
VALUES (value1, value2, value3...);
```

The second way is to insert values for every column in the table. If this is the case, you do not need to specify the column names, just the values:

```
INSERT INTO table_name
VALUES (value1, value2, value3...);
```

Specify Just Values

```
insert into sakila.actor  
values (202, "Kermit", "DaFrog", "2019-01-19 08:56:12");
```

Example 1: Specify Columns and Values

Consider the following query:

```
insert into sakila.actor (first_name, last_name, last_update)  
values ("Johnny", "Smith", "2019-01-17 08:43:11");
```

Example 3: Insert Data when Some is Missing

Sometimes, you want to insert a new row, but you don't have all the data for each column. If that is the case, there is an automatic fix for that. Consider below:

```
insert into sakila.actor (first_name, last_name)  
values ("Miss", "Pigee");
```

null

A column field that does not have a value will be read as a *Null* value. If a column allows for optional data, it is possible to insert a new record with no value. If this happens, then the record will be saved with a Null value. It is essential to understand that a Null value does not mean a value of zero or a field that contains spaces as its value. *Null* means that the value was left blank when the record was created.

It is possible to check using SQL keywords for Null values or non-Null values. You'll use the `IS NULL` and `IS NOT NULL` keywords in MySQL. Below is the syntax for `NULL` and `IS NOT NULL`:

NULL:

```
SELECT column_names FROM table_name  
WHERE column_name IS NULL;
```

IS NOT NULL:

```
SELECT column_names FROM table_name  
WHERE column_name IS NOT NULL;
```

example

What if you want to pull up everything that has multiple null values? At first, you might want to try something like this using the **AND** keyword:

```
select * from staff  
where picture and password is null;
```

But the above query will return nothing. If you look in the **staff** table, there is clearly an entry there fulfilling the criteria! Well, using the **AND** keyword is correct, but you need to check if each column is not null separately, like below:

```
select * from staff  
where picture is null and password is null;
```

The output of above will be this:

2	Jon	Stephens	4	Jon.Stephens@sakilastaff.com	2	1	Jon
2006-02-15 03:57:16							

You could also do something like:

```
select * from staff  
where picture is null and store_id = 2;
```

update

Update a Table

Now that you feel comfortable with Insert and Select, Update will be fairly straightforward. When updating, you have to use a new keyword: **SET**. This will set the field you want to be updated to the new data. Below is the syntax for **Update**:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

Go ahead and update the actor **Johnny Smith** that you inserted into the database earlier:

```
update actor  
set first_name = "Jonathan"  
where actor_id = 201;
```

Once the query is executed successfully, go ahead and view that specific customer using the below query:

```
select * from actor  
where actor_id = 201;
```

As you can see from the output, this customer's first name is now "Jonathan" instead of "Johnny".

example

```
update customer  
set active = 1  
where store_id = 2;
```

Columns

Defining the structure of your table is essential to storing your data in an efficient and meaningful way. *Columns* are the most essential part of the table, this is where the actual data will live. You will define these columns (also called fields), with a datatype that you deem appropriate for the data you wish to store. A good choice for a name, for instance, would be a `VARCHAR(50)`, with the number in the parentheses indicating the maximum number of characters that any given name may have. A good choice for storing a date of birth might be `DATE`, while `INTEGER` may be the best choice for the model year of a vehicle. Be sure to choose appropriately here, as changing any of these types could be a cumbersome undertaking if a significant amount of data has been stored in the table.

Nullability

To review what you have learned so far, `Null` is a value that is undefined. Because of this, attempts to compare values to a field that contains a `NULL` value will fail. The only operation that works on this value is the binary evaluation of `IS NULL` or `IS NOT NULL`. As applied to tables, this is whether a column in your table can contain the value `NULL` or not. The keywords are either `NULL`, or `NOT NULL` appended to the datatype that follows the column name when you create the table. For instance: `FirstName varchar(50) NOT NULL` would indicate that the `FirstName` column should never be undefined. A practical application of using NULLs would be something like a voluntary survey – perhaps the applicant chose not to answer ethnicity. So you could store `NULL` in that column when you save the data to indicate that there was no selection.

The structure of a table is critical. You will be working mainly with the rows and columns of the tables in a database. Tables can be related to Excel spreadsheets. The columns define the data needed, and the rows list out the data for each object in the database. Below is a MS Excel spreadsheet with information about specific animals in a pet store. Tables in a SQL database will be structured very similarly to the below MS Excel example:



	A	B	C	D
1	Name	Species	Breed	Age
2	Henry	Dog	Bulldog	2
3	Rupert	Pig	Mini	1
4	Churchill	Cat	Redpoint Siamese	3
5	Pepper	Dog	Mix	8

join

A frequent need is to combine data from multiple tables in a single result. Imagine a customer table and a separate order table. If you wanted to get all orders for a particular customer, you would need to join these two tables. In this lesson, we'll learn how to do just that.

A **JOIN** is an operand that allows you to select a row that contains columns from multiple tables. In the `sakila` database that you've been using, there are many tables, each with bits of information you might like to query.

By the end of the lesson, you will be able to:

- Join multiple tables to run a single query

The lesson will culminate in a Hands-On in which you will join multiple tables to run specific queries.

aliases

Aliases are used to give tables or columns different names than what is currently in the database. This is accomplished by using the `AS` keyword. Let's explore a scenario that could potentially require an alias.

You are working with a database, but someone else compiled the database and the column and table names. However, what you are doing with the data doesn't quite line up with the names of the columns for what you are trying to do. You could change the name of the columns using `AS` so it lines up better with your project. Consider below:

```
SELECT title AS filmTitle FROM sakila.film
```

innerjoin

When using `INNER JOIN`, it will select data that have matching values within two different tables. Below is the syntax for using `INNER JOIN`:

```
SELECT [columnName]
FROM [table1]
INNER JOIN [table2] ON table1.columnName = table2.columnName;
```

example

```
SELECT title, release_year, category_id
FROM sakila.film
INNER JOIN sakila.film_category
ON sakila.film.film_id = sakila.film_category.film_id
WHERE film.film_id = 1
```

```
SELECT customer_id
FROM orders
GROUP BY customer_id
ORDER BY COUNT(customer_id) DESC,
customer_id
LIMIT 1
```

multiple joins

Now that you have learned how to join two tables, you will learn how to join from multiple. You will use the same concept you learned from doing a singular join, except the main difference will be finding multiple columns that connect the tables together.

Look at the example below:

```
SELECT * from sakila.actor  
JOIN sakila.film_actor USING (actor_id)  
JOIN sakila.film_category USING (film_id)
```

In the query above, you are joining the `actor`, `film_actor`, and `film_category` tables together. First, you join the `actor` table and the `film_actor` table by using their common column, `actor_id`. After these two tables are joined, you are then able to join the `film_category` table due to the fact that the `film_actor` table had a common column (`film_id`) with the `film_category` table. Without joining the `film_actor` table to the `actor` table first, you would have not been able to join the `actor` table to the `film_category` table because they have no common columns.

outerjoin

Outer Joins

Having an inner join implies there is also an outer join, which is entirely correct. Furthermore, there is also a `left` and `right` variation. The difference between an inner and outer join has to do with when a row is selected to be included. Say there was a row in `albums` that was not linked to any rows in `tracks`. An inner join only includes the specified columns from a row if it can be paired up with a row in the joined table. So a row in `albums` that doesn't match any row in `tracks` would not show up at all in the results. Outer joins change this. An outer join takes every row from either the left or right (hence the `LEFT OUTER JOIN` or `RIGHT OUTER JOIN`) and *if a matching row exists in the other table*, then those columns are included as well.

example

A `LEFT OUTER JOIN` will return all records from the left table and all matched records from the right table. The result will be `NULL` on the right side if there is no match. The left table is the table name located right after the `FROM` keyword, and the right table is the table listed right after the `INNER JOIN` keywords.

Look at an example of the difference between an `INNER JOIN` and a `LEFT OUTER JOIN`:

```
SELECT * FROM sakila.film
INNER JOIN sakila.film_actor
ON sakila.film.film_id = sakila.film_actor.film_id
```

A `RIGHT OUTER JOIN` accomplishes the same thing as the `LEFT OUTER JOIN`, but it will be returning all of the records of the right table instead of the left.

Below is what you will get when you try to run a `RIGHT OUTER JOIN` on the same above queries:

```
SELECT * FROM sakila.film
RIGHT OUTER JOIN sakila.film_actor
ON sakila.film.film_id = sakila.film_actor.film_id
```

```
SELECT * FROM sakila.film
LEFT OUTER JOIN sakila.film_actor
ON sakila.film.film_id = sakila.film_actor.film_id
```

using

Great work so far! Now that you understand the basics of working with SQL Joins, you will learn a shortcut.

USING is a great way to cut down your query. You can use **USING** instead of **ON**. Consider the following query:

```
SELECT first_name, last_name, film_id  
FROM sakila.actor  
INNER JOIN sakila.film_actor  
ON sakila.actor.actor_id = sakila.film_actor.actor_id
```

We worked with this query earlier in this lesson. What this query is doing is joining the two tables using the **actor_id** column which both tables have. The **ON** statement is defining each table name then the column name we want (**sakila.actor.actor_id** or **sakila.film_actor.actor_id**). Since the column is the same name for both tables, you can use **USING**. Consider below:

```
SELECT first_name, last_name, film_id  
FROM sakila.actor  
INNER JOIN sakila.film_actor  
USING(actor_id)
```

walkthrou

```
SELECT rental_date, return_date, amount, title FROM sakila.rental  
JOIN sakila.payment USING (rental_id)  
JOIN sakila.inventory USING (inventory_id)  
JOIN sakila.film USING (film_id)
```

keyterm

```
SELECT actor_id, first_name, last_name,
       count(actor_id) as film_count
    from actor JOIN film_actor USING (actor_id)
   group by actor_id
  order by film_count desc
```

```
SELECT file_id, title, store_id, inventory_id
  FROM inventory
JOIN store USING (store_id) JOIN film USING (file_id)
 WHERE title = 'Academy Dinosaur' AND store_id = 1
```

and_or_not

The **AND** , **OR** , and **NOT** operators can be combined with the **WHERE** clause. The **AND** and **OR** operators are used to filter the data based on more than one condition. The **NOT** operator checks to see if something is not true. Look at examples of each.

syntax

The **AND** operator is checking to see if each condition separated by the keyword **AND** are all true.

```
SELECT [column-names] FROM [table-name]
WHERE condition1 AND condition2 AND condition3 ...;
```

The **OR** operator is very similar to the **AND** but its checking to see if a condition is true or another condition is true.

```
SELECT [column-names] FROM [table-name]
WHERE condition1 OR condition2 OR condition3 ...;
```

The **!=** operator is a way to check if a condition is not true. Check to see if the condition **RATING = "PG"** is not true;

```
SELECT title, rating FROM sakila.film
WHERE rating != "PG"
```

example_keyterm

in

The `IN` operator allows you to specify multiple values when using `WHERE`. It is shorthand for having multiple `OR` conditions.

There are two ways to use `IN`. The first is below:

```
SELECT [column_names] FROM [table_name]  
WHERE [column_name] IN (value1, value2, value3...);
```

And the second:

```
SELECT [column_names] FROM [table_name]  
WHERE [column_name] IN (SELECT statement);
```

Let's look at examples of both.

s

example

```
SELECT title, rating FROM sakila.film  
WHERE rating IN ("G", "PG", "PG-13")
```

Above, we are selecting the title and the rating from the film table where the rating's are "G", "PG", and "PG-13".

Example 2

The list that you're checking against doesn't have to be literal values and can even be the result of another query. See below:

```
SELECT title FROM sakila.film  
WHERE language_id  
IN (SELECT language_id FROM sakila.language WHERE name = "english")
```

like

Direct equality or numeric comparisons are not the only restrictions available. The **LIKE** operator is used in a **WHERE** clause to search for a particular pattern or character. When using **LIKE**, you need to use one of two wildcards:

- % (percent sign) represents zero, one or multiple characters
- _ (underscore) represents a single character

Below is the syntax for using **LIKE**:

```
SELECT [column-names] FROM [table-name]
WHERE [column-name] LIKE pattern;
```

example

Below will find any values that start with "h" and are at least three characters in length:

```
SELECT title FROM sakila.film  
WHERE title LIKE 'h_%_%';
```

Example 5

The below query will find any values that begin with "b" and end with "y":

```
SELECT title FROM sakila.film  
WHERE title LIKE 'b%y';
```

Example 6

Below will find any values that do NOT begin with "e":

```
SELECT title FROM sakila.film  
WHERE title NOT LIKE 'e%'
```

limit

`LIMIT` is a keyword that will limit the data to a certain number of rows. By using `LIMIT`, it prevents the query from returning an excessive amount of data, and in some cases that can be millions of rows.

An easy way to run a query with `LIMIT` is to right-click on the desired table and choose the "Select Rows (with limit)" option. That will automatically generate and execute the query. Try out this query:

```
SELECT title, description, release_year FROM sakila.film LIMIT 5
```

oder_by

Typically the rows in your table are not ordered in a particular way; in fact, it would be impossible to order them in every conceivable way you might want to retrieve the data. You can't order the tracks by Name and also by Milliseconds or by Bytes. Fortunately, SQL databases are very adept at doing this ordering for you during the request. All you have to do is add an **ORDER BY** clause to your query and specify which columns you want to use.

```
SELECT title FROM sakila.film  
ORDER BY title
```

By default, the ordering is done in ascending order. You should see the names going from smallest value to largest as determined by SQLite's rules for text order. If you want to reverse this, you can add **DESC** after the column.

```
SELECT title FROM sakila.film  
ORDER BY title DESC
```

ORDER BY can be combined with **WHERE** to make compound queries. Try finding all tracks less than 3 minutes in alphabetical order. See below for the query:

```
SELECT title FROM sakila.film  
WHERE length < 100  
ORDER BY title
```

select

When working in databases, we can manipulate the data in many ways. But before we can do that, let's learn how to read the data with which we are working. We do this with the **SELECT** keyword. The select statement is the single most important thing you need to understand in working with data in any SQL system. The most basic form of a select is:

```
SELECT [column_names] FROM [table_name];
```

upper_lower

The **UPPER()** function will convert all characters in a string to uppercase.

Run the following query:

```
SELECT UPPER("sql is super fun!") AS UpperCase
```

The **LOWER()** function will convert all characters in a string to lowercase.

Run the following query:

```
SELECT LOWER("I LOVE LEARNING SQL") AS LowerCase
```

where

```
SELECT [column names] FROM table_name WHERE [row restrictions]
```

Above, you will now include the **WHERE** keyword which will define any restrictions on the data.

Row restrictions are a check against the contents of one or more columns in the row. If the check succeeds, then the row is included in the result. Some very common restrictions have to do with equality and relative size. We can use symbols to check the equality and relative size. Below is a list of the common ones used:

- Equality: =
- Inequality: !=
- Greater than: >
- Greater than or equal to: >=
- Less than: <
- Less than or equal to: <=

Look at a real example with our database:

Say you want to find all films less than 90 minutes. After looking at the tracks table, you can see that you have a 'length' column. Now you can run a query to return all rows that are less than 90 minutes by running the below query:

```
SELECT * FROM sakila.film WHERE length < 100
```

indexes

Index	Indexes deal with the ordering of your data and have a significant impact on the performance of your database. They are used to retrieve the data from the database very quickly and improves the read time of the data.
CREATE INDEX	Used to create indexes in a table.
CREATE UNIQUE INDEX	This will create a unique index on a table. Duplicate values are not allowed.
EXPLAIN QUERY PLAN	Will show what is happening when a query is run.
min()	Will return the smallest value of the selected column.
max()	Will return the largest value of the selected column.
sum()	Will return the total sum of a numeric column.
avg()	Will return the average value of a numeric column.
count()	Will return the number of rows that match certain criteria.
GROUP BY	Is often used with the functions mentioned earlier in this lesson (MIN, MAX, SUM, AVG, COUNT). The results of these functions can then be grouped together using one or more columns.

clustered vs non

- **Clustered Index:** this is the actual, physical order of the data as it is stored on disk. When you have a clustered index, the SQL server stores it by that column. You can only have one of these on a table, and because of this, it is common that the primary key is also the clustered index. There are times you will want this not to be the case, but the vast majority of situations this will be your default. Think of it like a dictionary: the data is ordered in alphabetical order throughout the entire book, and when a new word is added, it is not added at the end, but where it belongs alphabetically.
- **Non-Clustered Index:** creates an entirely different object within the table. It contains the column or columns selected for indexing and points to the table's rows containing the data. The purpose of an index is to provide a way to expediently get your query results, without having to examine every row in a given table. The tradeoff of indexes is that when you create one, you copy all of the data you are indexing into a new structure with the order you have specified. Because of this, if there are too many indexes on a table, you may very well have far more storage dedicated to indexes than the actual storage of the table itself. It's not uncommon to see a table with many columns and 500mb of storage to have index space usage of over 1GB. Another drawback of too many indexes is that you may incur performance issues on heavily inserted/updated/deleted tables. That's because every time you do one of these operations – the same operation must be completed on every index to ensure data integrity. So index intelligently, and as infrequently as possible! A Non-Clustered index is like an index at the back of a book: keywords and references are stored there with the page numbers of the material for quick reference. Anytime the book is updated, the index needs to be updated as well.

Create index vs unique index

The CREATE INDEX statement is used to create indexes in a table.

The below query will create an index on a table. In this case, duplicate values are allowed:

```
CREATE INDEX index_name  
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX

This will create a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name  
ON table_name (column1, column2, ...);
```

Then, when you run a query on with the same specific columns you defined in the CREATE INDEX, it should pull in that data quicker than just running a query without an index. Make a note that your database isn't extremely large, so you may not see a considerable change in the speed of the data being pulled in. When you start working with a database with millions of rows, that's when indexes are vital and will help speed up the query.

Go ahead and run the following query:

```
CREATE INDEX filmListing  
ON sakila.film (film_id)
```

Maintaining Database

- Ensure that everyone who has access to the database requires access.
- Ensure default passwords are changed.
- Remove public access.
- Ensure that user groups / roles are assigned correctly. No one should be able to do more with the database than they need to.
- Divide up database administrative duties, so everything does not reside on just one person.
- Document your database structure and configurations.
- Create a testing environment, so that when you are making changes, you can double check they work before going live.
- Regularly do process checks on your data to make sure that everything is high quality and no breaches have occurred.
- Have a recovery plan for when disaster strikes.
- Know what to do when a data breach has occurred.

how to cmd

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~/IdeaProjects (master)
$ git config --global user.email "axwxred@gmail.com"
```

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~
git init|
```

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~
git commit -m "init commit"
```

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~
$ git branch -M 'main'|
```

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~/IdeaProjects (master)
$ git remote add origin https://github.com/Omarxploit/JavaDriverdemo.git
```

```
axwxr@LAPTOP-KP21UH88 MINGW64 ~
$ git push -u origin 'main'|
```

...or create a new repository on the command line

```
echo "# AnimalsSounds" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M 'main'
git remote add origin https://github.com/Omarxploit/AnimalsSounds.git
git push -u origin 'main'
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Omarxploit/AnimalsSounds.git
git branch -M 'main'
git push -u origin 'main'
```

Selenium

Help

<https://youtu.be/bpHTwO26HFc>

<https://www.springcloud.io/post/2022-03/spring-boot-integration-testing-mysql-crud-rest-api-tutorial/#gsc.tab=0>

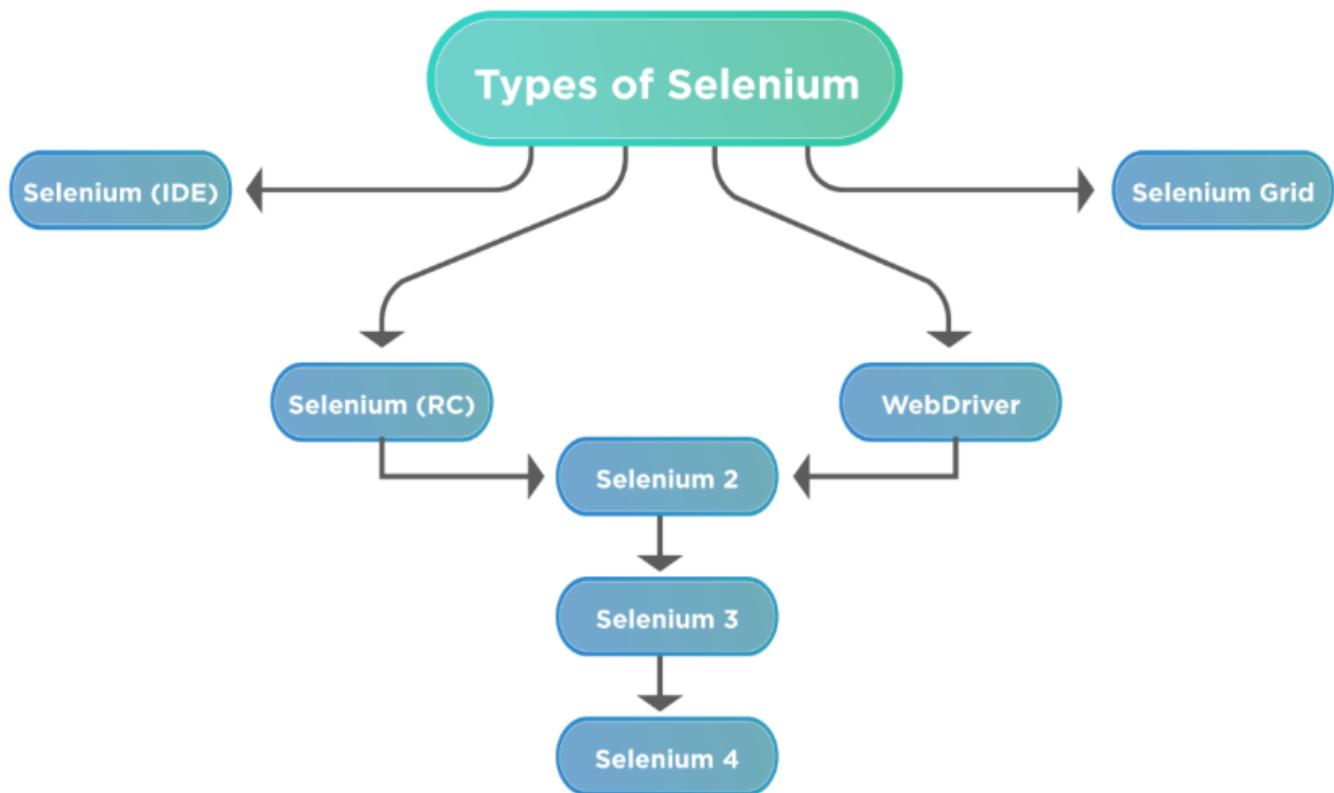
Basic understanding of Continuous Integration / Continuous Delivery concepts (CI / CD) and also should be aware of CI tools like Jenkins/Bamboo, etc.

SDETs are generally expected to take care of deployment issues as well hence understanding these tools is imperative.

They should be familiar with at least one front-end automation framework. The easiest and the most widely used is Selenium. It's the holy grail of front-end testing for web applications and almost all organizations use the Selenium framework for automating UI tests.

Introduction to selenium

Selenium is not just a single tool or a utility but is a suite of tools. Let's explore each tool to show how each tool is designed for a support testing environment. Selenium can be very confusing. If you are new to automated testing to automation testing using Selenium Confusion around Selenium (even though it is a prevalent testing framework) is well documented in the testing community. Be patient as you work through the tools; it may take some time to get acquainted with the entire suite.



<https://www.selenium.dev/selenium-ide/docs/en/introduction/command-line-runner>

Selenium IDE

Pros	Challenges
Ease of installation	Browser limits (only available in certain browsers)
No programming experience needed	Limited testing capabilities
Export tests into other Selenium projects	Iterations (or loops) structures not supported
Built-in tests results	
Robust suite of extension support	

Selenium Integrated Development Environment (IDE) is a plugin (originally Firefox) allowing testers to record their actions as they follow the workflow that they need to test. Selenium IDE has the capability of exporting code in various predefined languages. It also gives the ability to use one test case inside another. Selenium IDE is simple to use. Testers who may not fully understand any programming language can learn to use Selenium IDE. The ease of use of Selenium IDE is its primary advantage. However, Selenium IDE, because of its simplicity, does have limitations that are not appropriate for advanced testing.

Install

Install Selenium IDE Extension

Let's install the Selenium IDE. The installation of Selenium IDE is simple. It is available on the Google Chrome website as a browser extension.

1. To install Selenium IDE, navigate to the Google Chrome Web Store
2. Search for Selenium IDE
3. Choose "add to Chrome."

Using Selenium IDE

Now that you have Selenium IDE added, you can now use it to run tests. Once the extension is installed, you can start recording the tests. To execute a new test, follow the steps below.

We will start by

1. Click the Selenium icon found in your browser's toolbar.
2. Click on it to be taken to the point as shown below.

run test

Your interaction with the browsers is automatically recorded based on your "clicking" through the website. You will need to enter a valid URL of the site that you want to test before Selenium starts recording the interactions.

The screenshot shows the Selenium IDE interface. At the top, there's a search bar for 'Tests' and a URL input field containing 'http://www.woz-u.com'. A red arrow points from the text 'Enter a valid URL of the site you want to test' to the URL field. Below the URL field is a note: 'Note: The website will open in a new window'. Another red arrow points from the text 'Selenium records each action that you take on the site' to the test table. The test table has columns for 'Command', 'Target', and 'Value'. The recorded actions are:

Command	Target	Value
1 open	https://woz-u.com/	
2 set window size	1414x835	
3 click	id=input_12_1	
4 click	id=input_12_2	
5 type	id=input_12_2	
6 type	id=input_12_3	est
7 click	linkText=Cyber Security.	
8 assert not editable	css=.link-area-box-hover .content-box-heading	

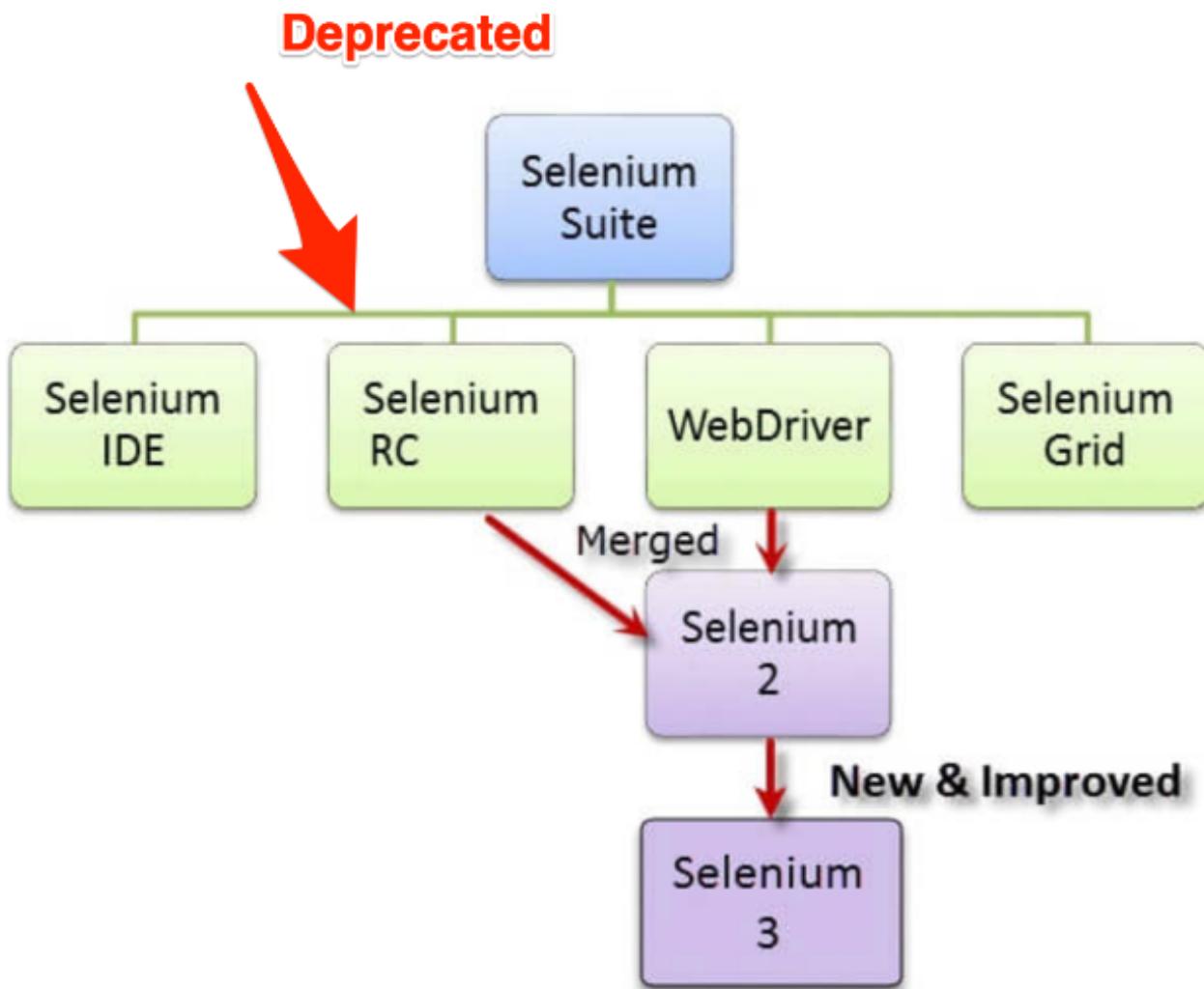
Note: The website will open in a new window

Selenium records each action that you take on the site

The interactions with the website (under test) are recorded and categorized into the following categories.

- Command- records the command executed by the tester (you)
- Target- records the target within the site
- Value- records the value of any input
- Description (Optional)

Selenium Remote Control (RC)



Even though Selenium IDE has many advantages, primarily being the ease of use, it does have its limitations. Where those limitations exist, Selenium Remote Control (RC) picks up where Selenium IDE leaves the testing space. Selenium Remote Control (RC) was the flagship testing framework. It makes full use of the full power of programming languages, such as Java, C#, PHP, Python, Ruby, and PERL to create more complex tests. Selenium allows automated web testing allowing programmers to use whatever language they prefer. In this tutorial, we will use Java. Currently, Selenium RC is deprecated and has been officially replaced by Selenium WebDriver. Here are some pros and cons of Selenium RC.

pro and cons

Cross-browser as well as cross-platform availability	Installation can be complex
No programming experience needed	Limited testing capabilities
Iteration support is available	Prerequisite knowledge is required (programming)
Faster execution than IDE	Relies on JavaScript
Supports data-driven testing	Very Complex Architecture

Selenium WebDriver



Selenium WebDriver is the successor to Selenium RC. WebDriver is actually a web automation framework. WebDriver has established itself to be more stable than both Selenium IDE and Selenium RC, which sends commands directly to the browser and retrieves results, implementing a more modern approach. Selenium WebDriver bypasses the need to rely on JavaScript for testing and communicates directly with the browser. It supports the following languages, although one does not have to know all of these languages:

- C#
- Java
- PHP
- Python
- Perl
- Ruby

Pros	Challenges
Simple Installation	Programming Knowledge required
Direct browser communication	Only standard (older browsers) are supported
Does not rely on a separate server (as RC does)	No real-time logging

Selenium Grid

We can use Selenium Grid to run parallel tests across different machines and different grid browsers simultaneously. This results in minimized execution time of automated tests. Selenium Grid takes advantage of Selenium RC for parallel test executions by utilizing the hub-and node concept. The hub acts as a central source of Selenium commands with each node connected to it.

Mobile Number or Email nitofe9687@weepm.com	<input checked="" type="checkbox"/>
Full Name johndou	<input checked="" type="checkbox"/>
Username johnn	<input type="button" value="X"/> <input type="button" value="↻"/>
Password abc123456	<input checked="" type="checkbox"/> Hide

Pros/Cons of Selenium WebDriver

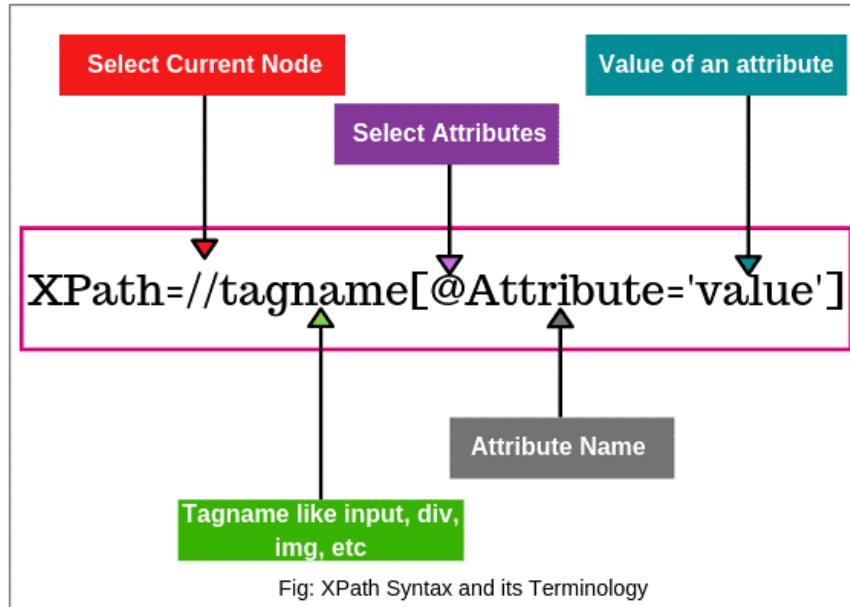
Pros	Challenges
Ease of installation	Only Supports Web Applications.
Open Source	Limited Support for Certain Features.
Solid Community Support	Built-In Reporting is Limited.
Easy to Implement	
Will Work on Multiple OSs	

Pros	Challenges
Flexible due to the number of ways it can be defined	Faster than Xpath
Can search for specific text using contains	Easy to read and supports all browsers
Can use complex selectors	Stable and robust
Can traverse the DOM tree using parent and sibling predicates	

Selenium Part2

As you have discovered earlier in your learning, Selenium can be a great tool for automated processes. As we will see in this lesson, there is quite a bit of expanded capabilities that a tool like Selenium can offer. Navigating to pages, interacting with various controls on the page, and seeing the outcome can help with various testing scenarios for specific inputs or some level of stress testing. By creating various scenarios, you can see the results and test out the logic much more quickly and easily than having to do something manually.

In this lesson, we'll take a deeper dive into Selenium. This will include interacting with web apps, parsing results, and scraping data. In addition, we will be interacting with various web controls by filling out and submitting forms. By making use of various HTML identifiers on the page, we can search for and retain the results from a search, from a form, or various other items on a page. As long as the element on the page can be identified, the data can be collected, analyzed, and retained.



access webdriver

Remember that, as you will be referring to it in our application later, like so:

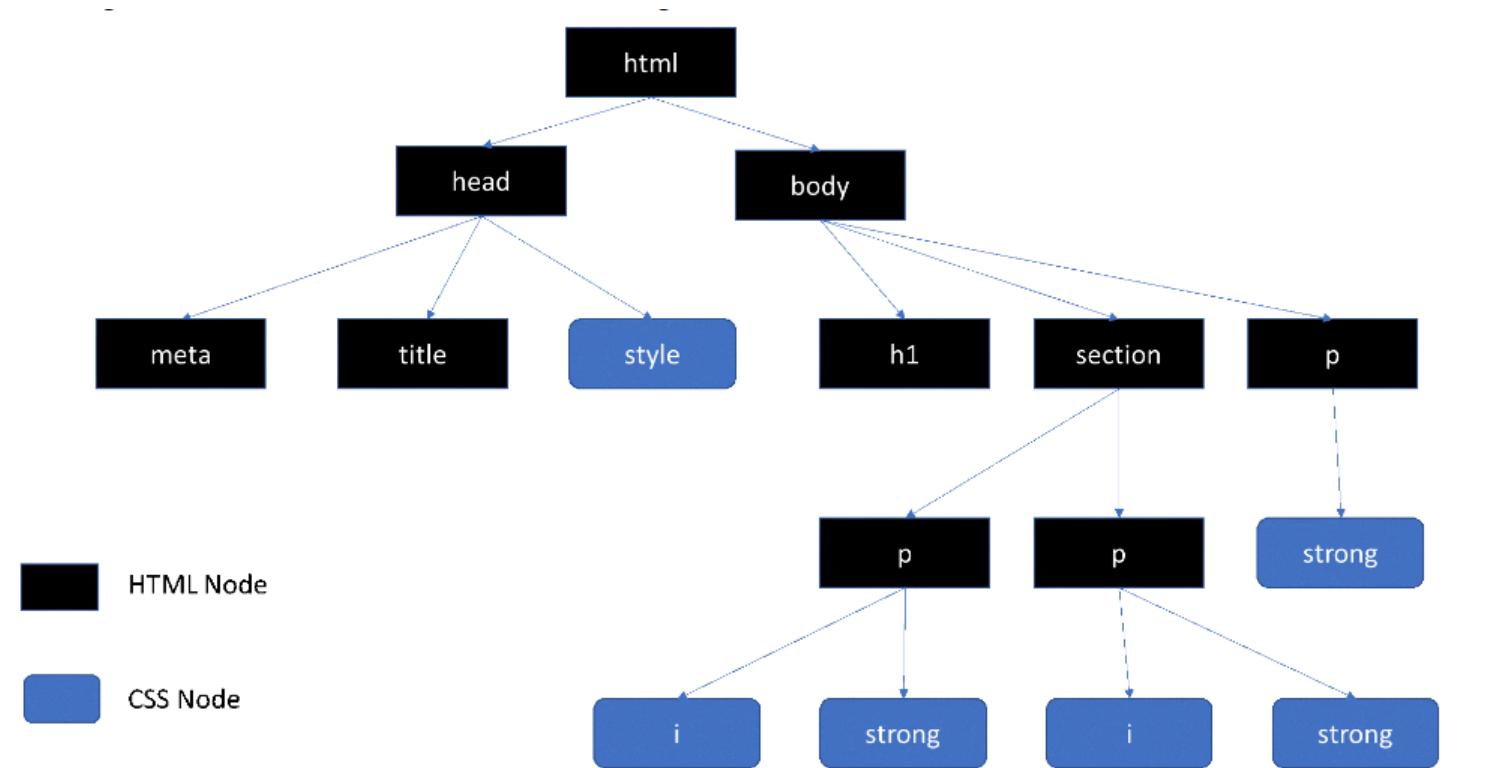
```
System.setProperty("webdriver.chrome.driver", "C:\\Users\\\\ Documents\\\\chromeDriver\\\\chromedriver.exe");
```

Dom (**D**ocument **O**bject **M**odel)

The DOM is the platform that a developer can use to access the content on a page. When a user requests a web page, that page is delivered to the requesting machine. Once that page and all of its associated assets are loaded, the browser creates a DOM of the page. The DOM is essentially a tree of objects.

In web parlance, those DOM objects can be accessed by Javascript, Jquery, or other web programming languages. In our example, we can also use Java to access the DOM and find/navigate the elements on the page. This is useful for gathering data, testing, and creating automated processes for looking through the content on a page.

HTML elements on a page almost always have either a class or ID that can be referenced by either CSS or JavaScript/JQuery for styling purposes and to potentially update page content. Using those same HTML tag attributes, you can write your application to access those same elements once the page has loaded.



Element Locators

As you begin to work on building an application that will interact with a website or web application, let's take a look at the actual page and what the intended outcomes are. A web page generally has some type of content. Sometimes that page will be interactive on either the client-side and/or will interact with the server-side to produce dynamic content. A static page will have content, menus, and other informational elements. You can read and review the content but there is no way to affect the page's content.

All of the elements on a page have locators, whether that is an element ID, field name, text, CSS selector, or Class. These attributes/data can be used to identify specific parts of a page. Using that in combination with Selenium, we can automate the process for testing applications, getting responses from forms, and collecting data from a page. Once it's collected, test outcome or data collection from a page can be stored in a database and analyzed at any point.

When a user interacts with those form elements (e.g. filling in data and asking questions), that data is sent to the server to be processed by both the server-side scripting language and, potentially, the database. When you search for a store location, a product, or any type of web form such as a contact form, e-commerce, or search form, all of the content is submitted to the server when the submit button is clicked. The server-side scripting language will then work with the data and, based on what type of form the data was sent from (e.g. contact form or shopping cart), the script will perform the appropriate processing. If data needs to be stored, this is the point at which it will be stored. Once all of the processing is done, there is generally a response that is sent to the user's browser, like search results or successful completion of their shopping needs.

Locators, Elements, and Wait

Locators are the tools that are used to find data, form fields, and essentially identify any of the items on a page. As you review your web applications, identifying the areas or the parts of the application you want to test allows you to designate which items will be collected and/or interacted with on a page.

ID

Name

Linktext

Tag Name

Class Name

CSS Selector

XPath

All of the above can be used to identify elements on a page. There can be times when not immediately attempting to find something on a page helps obtain the results you are after. Sometimes prompting the driver until the page finishes loading is important. That wait time can be either a set time or when the page has loaded.

Consider the following two methods for prompting the driver to wait until the page is loaded:

```
// Time wait method 1 a set number of seconds  
//         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

In the above method, you simply have the driver wait 10 seconds for the page to complete loading. The page could load quicker or slower sometimes, so this is not an accurate approach, but easier.

```
// Time wait method 2 until the page is loaded  
ExpectedCondition<Boolean> pageLoadCondition = new  
    ExpectedCondition<Boolean>() {  
        public Boolean apply(WebDriver driver) {  
            return ((JavascriptExecutor)driver).executeScript("return document.readyState").e  
quals("complete");  
        }  
    };  
  
WebDriverWait wait = new WebDriverWait(driver, 30);  
wait.until(pageLoadCondition);
```

In the above method, we have a boolean pageLoadCondition that is waiting for the page to return a document.readyState of complete or that it loaded. The driver will wait up to 30 seconds for this to occur. You could set the time longer, but 30 seconds is probably the max you would expect to wait.

The screenshot shows the Chrome DevTools interface with the 'Inspect Element' tool open. The element tree on the left shows a complex HTML structure with various divs, spans, and other tags. The context menu is open at the top right, with 'Relative XPath' highlighted under the 'Select' section. The right panel displays the selected element's CSS selector, which is //h1[@class='focus'].

On the above page, you can right-click on any element on the page and select inspect. There are browser plug-ins that will assist with this, as well, but even with the default inspect kit of the browser, you can find the various identifiers you need to make use of xpath in your Java project. Using the inspect that you've done and both the xpath method in Eclipse, along with the getText method, you can pull the content off of the page. Consider the following example:

```
driver.findElement(By.xpath("//h2[@class='focus']")).getText();
```

This will find on the page the H2 tag with the class focus and get the text displayed on the page for that element. Similarly, you could do the same thing with the ID element, as well. Later on, you will also take a look at looping through a group of elements with the same class or ID.

Conversely, you can also use CSS Selectors to interact/find content on a page, as well.

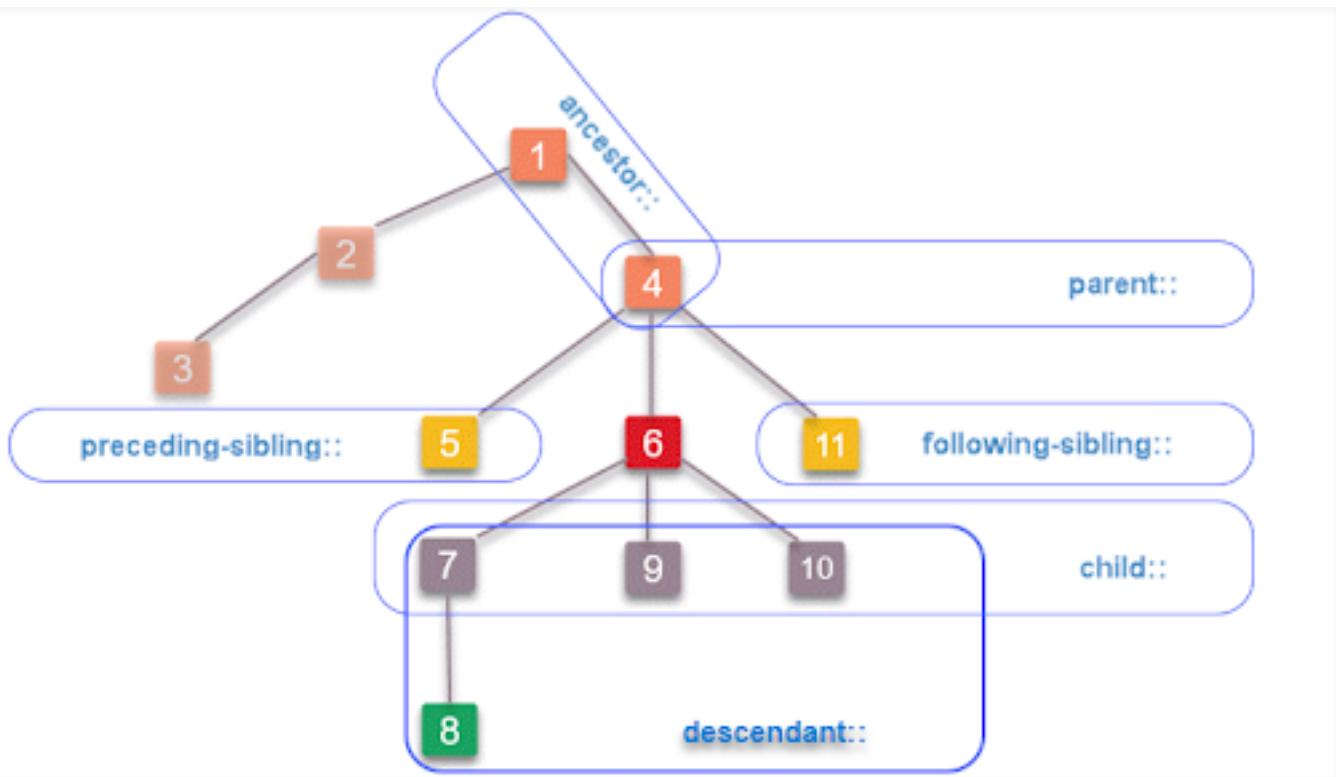
```
driver.findElement(By.cssSelector("#firstname"));
```

Duration.ofSeconds(10)

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Axes Methods and Forms

As we look at the code or inspect the various elements on a page, it is important to take a look at the various parent/child relationships throughout. The DOM helps to express the axes and the relationships that the hierarchical nature of HTML presents. XPath Axes can be used to find nodes relative to the node on the tree. Consider the following diagrams:



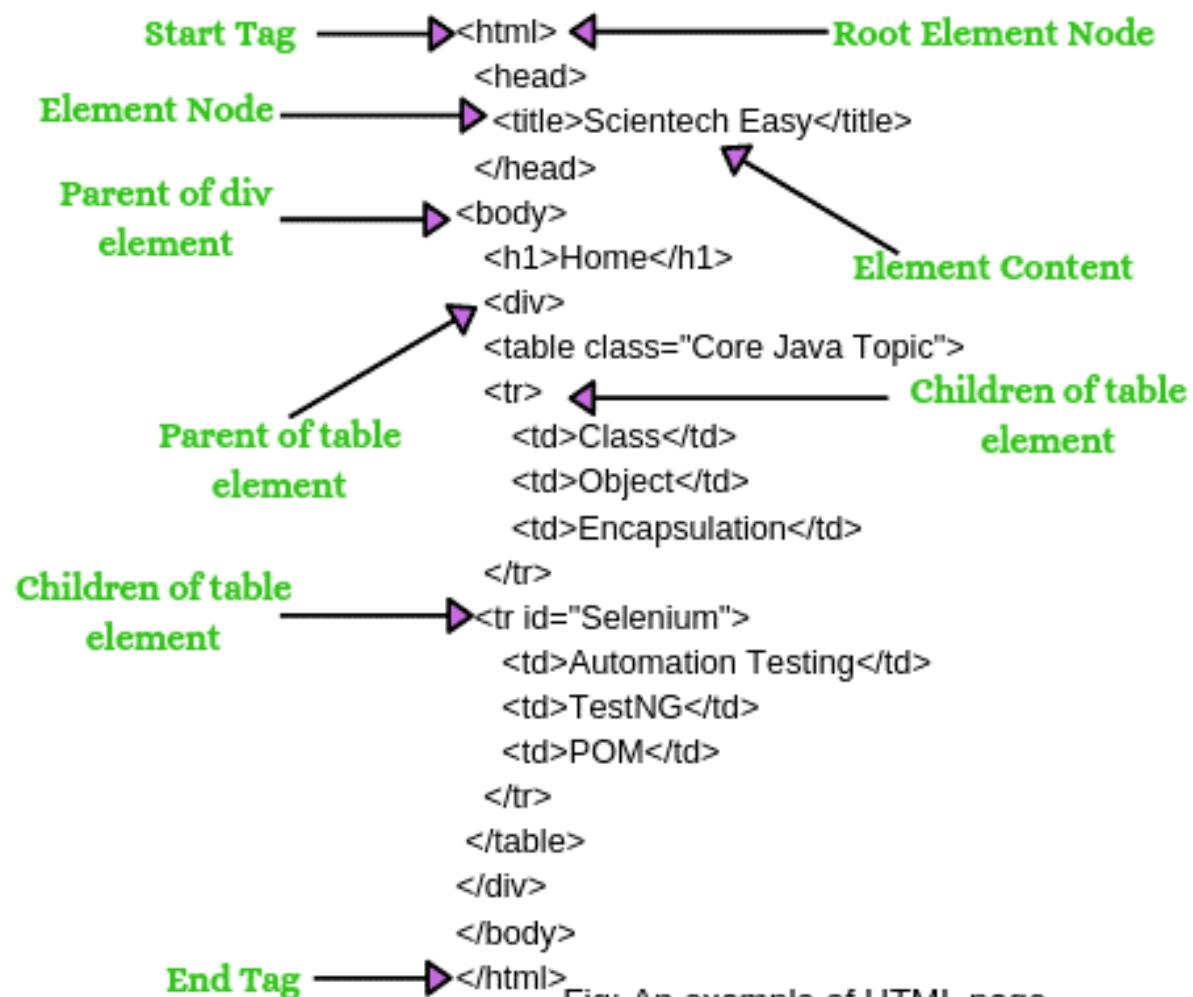


Fig: An example of HTML page.

Using our previous code for our last set of hands-on activities, remember our driver variable. Now imagine having a form field, like the one below, that you wanted to fill in. Using the *findElement* method and the *By class*, you can search the page for the form field Home Address or *hAddress*.

It is important to note that, as you make use of XPath, there are many syntax elements to help you write your statements to locate content on a page. Consider the slash and the double slash.

Using the single slash will function as an absolute XPath and starts the selection from the document start node. Using a double slash is a relative approach to XPath and starts from a point relative to the document.

Single Slash Syntax:

```
html/body/div[1]/div[2]/form/div[1]/div/input[1]
```

Multiple Slash Syntax:

```
//form/div[1]/div/div[1]/div/div/input[1]
```

In the above examples, single slash will start from the root, the beginning of the DOM model. This is somewhat slower than multiple slashes, but more complete and is less likely to break as the page content is updated. Multiple slashes are faster. Depending on the complexity and the amount of code that you have, those minor code changes can result in a decent amount of speed increases. As you develop your application, you will have to decide which trade-off is more valuable to you. Sometimes it can be one or the other.

```
<input type="address" placeholder="Home Address" name="hAddress" value="" class="form-control account-create-input-2 ">

driver.findElement(By.name("hAddress"));
```

Also, you can make use of XPath to find various dynamic elements on a page. You can look for specific elements on a page like:

```
Xpath=//input[@name='hAddress']
```

The above line of code will look for an input field with the name of hAddress. To set that value, you're going to use the sendKeys method to input a value into the form field.

Remember to create a webElement. In this case, you can call it emailAddress. WebElement emailAddress;

```
emailAddress.sendKeys("test@test.com");
```

Hidden Form Elements

Many forms have hidden elements that are used to pass data to another section of a form and are used for various processing purposes on the server-side when the form is submitted. Hidden data, while not used for any security or for sending sensitive information, is used for the logic and processing of visible form fields. Hidden form elements can also add context to the visible form data. You could include a user record number, a form page number, or other contextual data.

Hidden form fields have the input type of hidden when created and do not display in the browser. You can view the source of a page to see if there are any hidden fields. Attributes that are commonly used with a hidden form field are the name, id, and, of course, the value for that field.

With Selenium Webdriver, we can simply use the `sendKeys` method via `xPath` or you can create an object of `JavascriptExecutor` type. To use the `JavaScriptExecutor` method, you would also have to have the following import:

```
import org.openqa.selenium.JavascriptExecutor;
```

```
<script type="text/javascript">
<!--
if (typeof pid === 'undefined') {
  pid = Math.random() * 10000000000000000;
}
var idn = Math.random() * 10000000000000000;
document.write('<'+'c'+ip+'t type=' + "text/javascript" + ' src="http://101-beads.newsyclecloud.com/apps/0AMS.dll/src/be801/CRAN00/CRAN/' + pid + '/-1/-;idn=' + idn + ';Type=1?></script>');
//-->
</script>

</div>
<!-- HOUSE - NEWSLETTER -->
<!-- Begin CTCT Sign-Up Form -->
<div rel="stylesheet" type="text/css" href="https://static.ctctcdn.com/h/contacts/embedded-signup-assets/1.0.2/css/signup-form.css">
<div class="ctct-embed-signup-widget shadowed" style="width: 300px; -webkit-font-smoothing: antialiased;">
  <div style="padding: 0; margin: 0; width: 290px;">
    <span id="success_message" style="display:none;">
      <div style="text-align:center;">Thanks for signing up!</div>
    </span>
    <form data-id="embedded_signup_form" style="padding:0; margin: 0; width: 290px;" name="embedded_signup" method="POST" action="https://visitor1.constantcontact.com/api/signup">
      <h3>Get Daily <b>and Breaking News Updates!</b></h3>
      <div class="underline"></div>
      <p>Get the latest news from The Cranberry Eagle right to your inbox. To finish signing up, click Sign Up.</p>
      <!-- The following code must be included to ensure your sign-up form works properly. -->
      <input data-id="email" type="hidden" name="ctct_email" value="1_REPLACE_1">
      <input data-id="submit" type="hidden" name="submit" value="Sign Up" style="width: 100%; height: 35px; border: none; border-radius: 5px; background-color: #0070C0; color: white; font-size: 14px; font-weight: bold; padding: 0 10px; margin-top: 10px;">
      <input data-id="urlinput" type="hidden" name="url" value="https://www.ctctcdn.com/h/contacts/embedded-signup-asset/1.0.2/css/signup-form.css?pid=1&idn=1&Type=1" style="width: 100%; height: 35px; border: none; border-radius: 5px; background-color: #0070C0; color: white; font-size: 14px; font-weight: bold; padding: 0 10px; margin-top: 10px;">
    </form>
  </div>
</div>
```

```
By hiddenObj = By.xpath("//*[@id='idOfYourHiddenElement' and @type='hidden']]");
```

```
JavascriptExecutor jScript = (JavascriptExecutor)driver;
jScript.executeScript( script: "document.getElementsByName('_mc4wp_timestamp').value='1234567890'" );

JavascriptExecutor hScript = (JavascriptExecutor)driver;
hScript.executeScript( script: "document.getElementsByName('_mc4wp_form_element_id').value='mc4wp-form-2'" );
```

Windows & iFrames

There are times where content is loaded into a page/site from other domains or other pages. At times, you might want to interact with that content to either gather information or to interact with form objects in the *iFrame*. The *iFrame* tag is simply stated, to load or embed content with the current page.

You might see this to embed third-party content, like a video, PDF, or other types of content not native to that site or page. This tag could also be used to scroll through a lot of content, like a policies page for instance. Instead of the user having to lose their place on a page, the iframe could be used to review policies or other types of materials. It could be useful in a testing scenario to get that content and analyze the information.

Iframe Example

The following demonstrates an *iframe* with basic attribute settings:

```
<iframe id="ifrm" src="demo.html"></iframe>
```

The *iframe*'s *src* attribute specifies the URL of the document to be displayed in the *iframe*. Although attributes are available to control *width*, *height*, *scrolling*, and *frameborder*, we use the following CSS to control the appearance of our example:[1]

```
iframe {  
    border:1px solid #ccc;  
    width:80%; height:120px;  
}
```

This is how the *iframe* displays:

Iframe Demo

The *iframe* content comes from a separate HTML document. It has its own style sheet. It can also contain script.

Scrollbars will appear automatically if they are needed to display all the content in the *iframe*. If you don't want the *iframe* scrollbars to appear, you can use JavaScript to set the *height* of the *iframe* to show all its content.

[Find out how to set up links to load new documents into the *iframe*.](#)

Information on Demo

[Introduction to Iframes](#)

More Iframe Topics

[Iframes Tutorial: Overview](#)

[Changing Iframe Src](#)

[Setting Iframe Height](#)

[Cross-Document Communication](#)

[Iframe Onload Event](#)

[And More...](#)

Get Code

[Download Examples](#)

Clearance Today on New & used

Chrysler, Jeep, Dodge Ram
Be treated like family at Napleton Ellwood city Chrysler, Jeep, Dodge Ram Truck dealership

[OPEN](#)

Interacting with Content in an iFrame

In this hands-on activity, you're going to load the page and get the course of the iFrame. Remember, iFrames can load several different types of content, so consider how you might interact with that content.

You're going to switch to the frame and get the content, similar to the following:

```
driver.switchTo().frame("the_frame"); // remember to find the name of the frame
```

```
System.out.print(driver.getPageSource()); //used to output the content of the iframe
```

1. Create a new project in Eclipse and import the test script below.

Here is a quick refresher on how to do that:

1. Open Eclipse and create a new **Java Project** named `WebDriverPracticeHandsOn`.
2. In the **Package Explorer**, right-click `src` and create a new **Class** with the following information:
 - o **Package:** "com.your-name.handsOn1" (or *default*)
 - o **Name:** "FirstScript"
 - o Ensure the `main` method stub is **checked**.

2. Add the elements listed above like:

```
driver.switchTo().frame("the_frame");
```

```
driver.switchTo().frame("ifrm");
driver.getPageSource();
System.out.print(driver.getPageSource());
```

```
WebElement iFrame = driver.findElements(By.tagName("iframe")).get(1);
driver.switchTo().frame(iframe);
driver.getPageSource();
driver.switchTo().defaultContent();
```

JavaScript

And:

```
JavascriptExecutor jScript = (JavascriptExecutor) driver;
```

You can begin to make use of the executeScript method to implement your JavaScript.

```
jScript.executeScript("document.getElementById(' element id').click();");
//or
jScript.executeScript("arguments[1].click()", login);
```

2nd Paragraph in this section: Javascript is a client-side scripting language used in web development. Client-side scripting languages provide for interactivity and many of the dynamic elements that you see and interact with on a page. That pop-up box, that drop-down menu, and any other number of interactive elements on a page are provided for by JavaScript.

```
<a href="#" onclick="return false;">Go</a>
<a href="javascript:void(0);">Go</a>
<a href="#" id="a">Go</a>
<script>
    document.getElementById("a")
        .addEventListener("click",function(e){
            e.preventDefault();
            return false;
        },false)
</script>
```

xpath vs js

XPath	JavaScriptExecutor
Complex selectors allow for multiple combinations	Native to all browsers and the DOM
Can search for specific text using contains	Used by early versions of Selenium
Can traverse the DOM tree using parent and sibling predicates whereas	Learning curve is less steep to learn JavaScript
Contains many built-in functions	

keyterm

XPath	XML path for navigating through a page's structure
By	Object used to locate items on a page by class, id, name, or other HTML tag attributes
WebElement	A DOM (Document Object Model) element used to find HTML tags on a page using id, name, class, xpath, CSS selectors, link text, and other elements.
WebDriver	The main element in Selenium used to automate the testing of a web application or set of pages. WebDriver will allow for control of the browser, selection of various web elements, and general debugging exercises.
SendKeys	A method in Selenium used for entering values into various editable fields on a page.
JavascriptExecutor	An interface in Selenium used to actuate JavaScript. Things in JavaScript, like getElementbyID and many other JavaScript tools, are available to be used in conjunction with Selenium. This makes interacting with pages much more native and easy.
findElement	A method used to identify elements on a webpage. Usually used in conjunction with the By Object and can find items on a page by ID, name, class name, tag name, link text, or even partial link text.
executeScript	Method used to execute JavaScript on a page. Is called through the JavaScriptExecutor import.
getText	Used to get the value of a web element and will return a string.

what is devOP

The DevOps movement has produced several principles that have evolved over time and are still evolving. Several solution providers, including IBM, have developed their own variants. All these principles, however, take a holistic approach to DevOps, and organizations of all sizes can adopt them. These principles are:

- ✓ Develop and test against production-like systems
- ✓ Deploy with repeatable, reliable processes
- ✓ Monitor and validate operational quality
- ✓ Amplify feedback loops

At its root, DevOps is a cultural movement; it's all about people. An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools. .

how to build it

Building a DevOps culture, therefore, is at the core of DevOps adoption. A DevOps culture is characterized by a high degree of collaboration across roles, focus on business instead of departmental objectives, trust, and high value placed on learning through experimentation.

TECHNIQUES

Following are a few specific techniques that you need to include when you adopt DevOps:

- ✓ Continuous improvement
- ✓ Release planning
- ✓ Continuous integration
- ✓ Continuous delivery
- ✓ Continuous testing
- ✓ Continuous monitoring and feedback

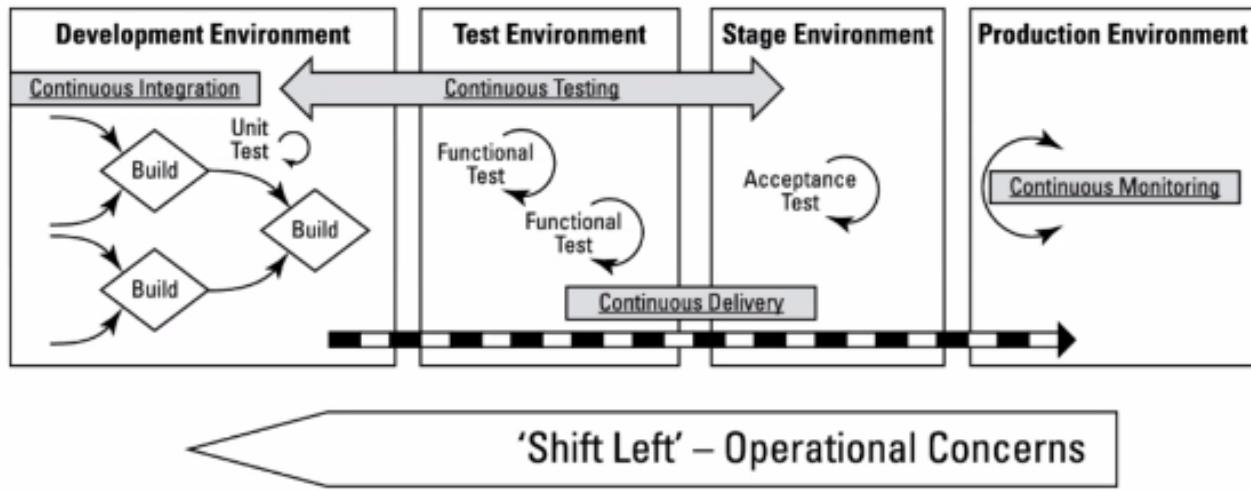


Figure 1-1: The shift-left concept moves operations earlier in the development life cycle.

github

[GitHub](#) creates an environment that allows you to store your code on a remote server, gives you the ability to share your code with other people, and makes it easy for more than one person to add, modify, or delete code to the same file and project, while keeping one source of truth for that file (phew!). So what does that all actually mean? One of my favorite ways of explaining GitHub.com to folks who are new to the tool is to compare it to Google Docs — a place online where you can write code with other people and not have to email different versions back and forth.

Introduction to Deployment

Deployment is the act of uploading an application to a server. Frequent deployment of an application is quite common, as each deploy may contain an updated web application with an added feature or security update. Deployment is composed of several components that work in conjunction with one another.

- **Server:** The server provides the infrastructure for communication via the web, making it possible to respond to requests from clients. In this course, the server will be using an Amazon service called EC2.
- **Repository:** A repository host such as GitHub is commonly used among development teams. The benefits of using a shared repository extend beyond improving the development process. This empowers the team to have one workspace for testing code and another workspace for *production*, or code that is ready to be used by real users.
- **Continuous Integration:** It's tedious for a developer to manually test the code, zip the code into a file, and deploy the code. Not to mention, if the developer forgets to test one component, it could be a critical bug that renders the application useless until it's patched. Fortunately, some tools do this process of *continuous integration* on behalf of the developer. In this course, the continuous integration tool will be Travis CI.

Starter Application

1. You are now going to fork the `hackathon-starter` project. Be sure to note that *forking* is different from *cloning*. Cloning will copy the project onto your local machine and forking will make a copy on your GitHub account and add it to the list of your repositories.

 Navigate to the `hackathon-starter` repo on GitHub:

[Hackathon-Starter on Github](#)

 The goal here is to make a copy of this repo and work on changes independent of the original repo. This can be accomplished by clicking the `fork` option in the top-right corner of the page as shown below:

2. **Clone** the project you just forked onto your local machine's desktop:

- Do NOT clone the original project. You want to clone the project on your GitHub account.
- `cd desktop`
- `git clone githubUrl`
 - make sure to use the https URL to the hackathon-starter located on **your** GitHub account as the `githubUrl`.

Then, `cd` into the `hackathon-starter` project located on your desktop.

Run `npm install`

- If Node or npm are not installed, download [Node.js and npm](#).

Now that the application has been built on your machine, it's time to provide a MongoDB database for the application to operate. If you do not have the MongoDB server downloaded, it can be downloaded at:

[MongoDB Download](#)

Click on "Community Server" and download the current stable release for your operating system (Mac or Linux).

In a new terminal window, `cd` into the downloaded Mongo server folder.

- You may have to `cd` into downloads first, then the Mongodb folder that was downloaded.
 - The Mongodb folder may be named something like `mongodb-osx-x86_64-3.4.10` on a Mac.

Now, `cd` into `bin` that lives within the Mongo folder.

Run `mkdir -p data` to create a folder within the `bin` folder.

Next, run `./mongod --dbpath ./data` to start up Mongo

- At this point, the dependencies for the application have been installed successfully, and MongoDB is running.

code analysis

In this section, we'll review code analysis and how it fits into the DevOps process. With the adoption of quick-moving programming methods such as Agile, DevOps, and continuous integration, there is real concern that the quality of the code created could suffer from the fast-moving timelines. While the benefits of these adoptions are obvious, such as the latest features, fixes, and security being provided to customers as soon as they are ready, are there also additional bugs and quality control issues being introduced?

Code Checks Prior to Deployment

To prevent bugs and security vulnerabilities from being introduced, it is possible to implement forms of static code analysis that would automatically scan the code for vulnerabilities or weaknesses during development and prior to deployment. If vulnerabilities or errors are detected in the code after deployment, the central repositories are enabled to deny the code commits in order to prevent these vulnerabilities from being introduced. This static analysis can also be performed in the *integrated development environment (IDE)* as well to further enhance the detection and mitigation of vulnerabilities.

This type of testing requires automation, as manual testing would simply be unable to meet the velocity requirements where it really matters for the average DevOps, Agile, or CI/CD methods. This is where static code analysis begins to show its value. By bringing in a suite of automated checks that are capable of scanning for compliance issues and security vulnerabilities, it reduces a significant amount of overhead for manual verification and code review. In most cases, common security vulnerabilities are well known and documented, which makes it easy to create automated filters and checks to indicate detected issues. Compliance requirements can be programmed into the code analysis tool so there can be a focused approach on the more complex compliance and security issues that aren't so easily automated. This frees up resources to tackle the hardest issues and provide more quality code to the end users.

On top of automating the security tests, it is possible to develop automated tests that check application functionality and acceptance to ensure that the application is operating efficiently even after code deployment. There are quite a few different types of automated testing that can be performed:

Checks Prior to Deployment

- Unit Tests
 - Coded verifications that are able to validate specific behaviors in a focused area within a system. Consider a system that calculates wages based on an hourly rate, it would need to validate standard hours, overtime, and any errors such as negative time, or wages, or those over the maximum hour limit.
- Integration/Service Tests
 - Most often designed and developed by the developers of the application themselves, these tests check behaviors of the application such as web services, database calls, and other API interactions.
- Functional Tests
 - Do portions of the application still function? Ask questions such as: "Can wages still be submitted? Can users still access the interface? Are there any errors when entering data?"
- Performance
 - Answers the question, "Can the system operate efficiently without slowing down and impairing user operation?"
- Load
 - Load tests would be automated to ensure that the application can handle the expected usage. A website and server might be tested with an automated system to see how many concurrent users can be active, or a test could be performed to see if the system can handle a certain number of interactions.
- Security
 - These tests would take into account the security and vulnerability assessments we discussed previously. These would automatically find the detectable errors to free up resources for more intensive investigations and development.

Accessibility

- Tests that check to see if all parts of the program are accessible and if they require any specific augmentations for usage by their target audience.

Production monitoring

- Tests that check whether the application achieves the output expected.

Azure DevOps

In this section, we'll discuss Azure DevOps. Microsoft offers a software as a service (SaaS) platform for DevOps, called Azure DevOps. This tool is designed to provide an end-to-end DevOps toolchain for teams to effectively and rapidly develop and deploy software. It integrates with most leading tools that are currently available and provides a great architecture for orchestrating and managing a DevOps approach.

There are several main services offered by Azure DevOps, which are:

- Azure Boards
 - Focused around delivering value to users through proven agile tools that enable planning, tracking, and discussion work across the development teams.
- Azure Pipelines
 - Build, test, and deploy with CI/CD that works agnostic of the language, platform, and cloud. This service can interact with GitHub or any other Git provider and deploy continuously.
- Azure Repos
 - Providing unlimited, cloud-hosted private Git repos and collaborative environments, the Repos enable better code to be built with pull requests and advanced file management.
- Azure Test Plans
 - Test and ship with confidence using manual and exploratory testing tools.
- Azure Artifacts
 - Create, host, and share packages within the teams, and add artifacts to the CI/CD pipeline with a single click.

With the end-to-end solutions on Azure, teams can implement the necessary DevOps practices to each of the application lifecycle phases: plan, develop, deliver, and operate. These DevOps technologies, combined with people and processes, enable teams to continually provide value to customers.

- Plan
 - Empower teams to manage their work with agility and full visibility across products and projects. Define, track, and lay out work with Kanban boards, backlog custom dashboards and reporting capabilities using Azure Boards. Keep development efforts transparent and on schedule with GitHub. Explore analytics with visuals and turn data into insights with Power BI.
- Develop
 - Code faster and smarter with Visual Studio and Visual Studio Code. Share code and collaborate with like-minded developers with GitHub. Automate testing and practice continuous integration in the cloud with Azure Pipelines, create automatic workflows from idea to production with GitHub Actions, and even bring your Jenkins workloads to Azure. Provision environments for developers in minutes with Azure DevTest Labs.
- Deliver
 - Deploy your application to any Azure service automatically and with full control to continuously deliver value to customers. Define and spin up multiple cloud environments with Azure Resource Manager or HashiCorp Terraform, then create continuous delivery pipelines into these environments using Azure Pipelines or tools such as Jenkins and Spinnaker.

Operate

- Implement full stack monitoring, get actionable alerts, and gain insights from logs and telemetry with Azure Monitor. Manage your cloud environment with Azure Automation and tools such as Ansible, Chef, or Puppet.
- Keep provisioned infrastructure and applications in compliance with Azure Blueprints or Chef Automate. Used with Azure Security Center, you'll limit threat exposure as well as find and remediate vulnerabilities quickly. [\[source\]](#)

Automation

The Azure DevOps Automation platform provides a cloud-based automation and configuration service that enables teams to consistently deliver across the Azure and non-Azure environments. It constitutes process automation, configuration management, update management, shared capabilities, and heterogeneous features. It seeks to provide complete control during deployment, operations, and decommissioning of workloads and resources.

Process automation

- Orchestrate processes using graphical, PowerShell, and Python runbooks.

Configuration management

- Collect inventory
- Track changes
- Configure desired state

Update management

- Assess compliance
- Schedule update installation

Shared capabilities

- Role-based access control
- Secure, global store for variables, credentials, certificates, connections
- Flexible scheduling
- Shared modules
- Source control support
- Auditing
- Tags

Heterogeneous features

- Windows and Linux
- Azure and on-premises

Process Automation

The primary focus with process automation is to remove the need for resources to perform frequent, time-intensive, and error-prone management tasks in the cloud. By removing this requirement, it is possible to focus on the work that drives value to the business. The reduction in errors and boost in efficiency enables organizations and teams to reduce their overall operational costs and bring in savings. Process automation supports the integration of Azure services and other public systems required for deploying, configuring, and managing your end-to-end processes. The service allows you to author runbooks graphically, in PowerShell, or using Python.

Update Management

Included in the Azure Automation, this Update management solution is capable of administering Windows and Linux systems across hybrid environments to enhance visibility of update compliance across Azure and other clouds, and on-premise. There are options to schedule deployments of orchestrated updates within specified maintenance windows, you can even exclude updates from installation to specified machines.

Shared Resources

Azure Automation consists of a set of shared resources that make it easier to automate and configure your environments at scale.

- **Schedules**--Trigger Automation operations at predefined times.
- **Modules**--Manage Azure and other systems. You can import modules into the Automation account for Microsoft, third-party, community, and custom-defined cmdlets and DSC resources.
- **Modules gallery**--Supports native integration with the PowerShell Gallery to let you view runbooks and import them into the Automation account. The gallery allows you to quickly get started integrating and authoring your processes from PowerShell gallery and Microsoft Script Center.
- **Python 2 packages**--Support Python 2 runbooks for your Automation account.
- **Credentials**--Securely store sensitive information that runbooks and configurations can use at runtime.
- **Connections**--Store name-value pairs of common information for connections to systems. The module author defines connections in runbooks and configurations for use at runtime.
- **Certificates**--Define information to be used in authentication and securing of deployed resources when accessed by runbooks or DSC configurations at runtime.
- **Variables**--Hold content that can be used across runbooks and configurations. You can change variable values without having to modify any of the runbooks or configurations that reference them.

Role-Based Access Control

Regulates access and control across the Automation account and its resource utilizing *role-based access control (RBAC)*.

Source Control Integration

Heterogeneous Support (Windows and Linux)

Automation is designed to work across your hybrid cloud environment and also your Windows and Linux systems. It delivers a consistent way to automate and configure deployed workloads and the operating systems that run them.

Common Scenarios for Automation

Azure Automation supports management throughout the life cycle of your infrastructure and applications. Common scenarios include:

- **Write runbooks**--Author PowerShell, PowerShell Workflow, graphical, Python 2, and DSC runbooks in common languages.
- **Build and deploy resources**--Deploy virtual machines across a hybrid environment using runbooks and Azure Resource Manager templates. Integrate into development tools, such as Jenkins and Azure DevOps.
- **Configure VMs**--Assess and configure Windows and Linux machines with configurations for the infrastructure and application.
- **Share knowledge**--Transfer knowledge into the system on how your organization delivers and maintains workloads.
- **Retrieve inventory**--Get a complete inventory of deployed resources for targeting, reporting, and compliance.
- **Find changes**--Identify changes that can cause misconfiguration and improve operational compliance.
- **Monitor**--Isolate machine changes that are causing issues and remediate or escalate them to management systems.
- **Protect**--Quarantine machines if security alerts are raised. Set in-guest requirements.
- **Govern**--Set up RBAC for teams. Recover unused resources.

AutomationThe Azure DevOps Automation platform provides a cloud-based automation and configuration service that enables teams to consistently deliver across the Azure and non-Azure environments. It constitutes process automation, configuration management, update management, shared capabilities, and heterogeneous features. It seeks to provide complete control during deployment, operations, and decommissioning of workloads and resources.

Process automationOrchestrate processes using graphical, PowerShell, and Python runbooks.

Configuration managementCollect inventory

Track changes

Configure desired state

Update management

Assess compliance

Schedule update installation

Shared capabilities

Role-based access control

Secure, global store for variables, credentials, certificates, connections

Flexible scheduling

Shared modules

Source control support

Auditing

Tags

Heterogeneous features

Windows and Linux

Azure and on-premises

Process AutomationThe primary focus with process automation is to remove the need for resources to perform frequent, time-intensive, and error-prone management tasks in the cloud. By removing this requirement, it is possible to focus on the work that drives value to the business. The reduction in errors and boost in efficiency enables organizations and teams to reduce their overall operational costs and bring in savings. Process automation supports the integration of Azure services and other public systems required for deploying, configuring, and managing your end-to-end processes. The service allows you to author runbooks graphically, in PowerShell, or using Python.

Configuration ManagementThe Azure Automation State configuration provides a PowerShell desired state configuration (DSC) service for enterprise environments that enables the management of DSC resources in Azure automation to apply configurations to virtual or physical machines. This tool is capable of monitoring the update configurations across all machines, virtual or physical, Windows or Linux, in cloud or on-premise. There are also features that can query systems to determine resource utilization, installed applications, and other configuration items.

Update ManagementIncluded in the Azure Automation, this Update management solution is capable of administering Windows and Linux systems across hybrid environments to enhance visibility of update compliance across Azure and other clouds, and on-premise. There are options to schedule deployments of orchestrated updates within specified maintenance windows, you can even exclude updates from installation to specified machines.

Shared ResourcesAzure Automation

consists of a set of shared resources that make it easier to automate and configure your environments at scale.**Schedules--Trigger Automation operations at predefined times.****Modules--Manage Azure and other systems.** You can import modules into the Automation account for Microsoft, third-party, community, and custom-defined cmdlets and DSC resources.**Modules gallery--Supports native integration with the PowerShell Gallery** to let you view runbooks and import them into the Automation account. The gallery allows you to quickly get started integrating and authoring your processes from PowerShell gallery and Microsoft Script Center.**Python 2 packages--Support Python 2 runbooks for your Automation account.****Credentials--Securely store sensitive information that runbooks and configurations can use at runtime.****Connections--Store name-value pairs of common information for connections to systems.** The module author defines connections in runbooks and configurations for use at runtime.**Certificates--Define information to be used in authentication and securing of deployed resources when accessed by runbooks or DSC configurations at runtime.****Variables--Hold content that can be used across runbooks and configurations.** You can change variable values without having to modify any of the runbooks or configurations that reference them.**Role-Based Access Control**

Standardization

Standardization in the DevOps approaches will likely encounter several challenges in order to implement an effective and streamlined process. There is no centralized group for DevOps as it is a group of concepts and practices that seek to combine software development and information technology operations with the aim of reducing the systems development cycle and providing continuous delivery with high software quality. This leads to multiple interpretations and various implementations that end up being situational to the organization and its internal mobility.

Azure DevOps helps address some of the primary challenges that teams may encounter when deploying DevOps:

- Tool challenges:
 - Teams face a need for multiple tools with varying features, limited knowledge, and the ability to integrate with their software pipelines.
- Azure DevOps provides a robust toolset within their cloud platform enabling the teams to work together collaboratively and access documentation to increase knowledge of their tools.
- Process challenges
 - Without a central DevOps team, recommendations for tool integrations into the processes, lack of KPI or SME knowledge, a team can run into challenges implementing the process to standardize the development process.
- Azure DevOps platform and tools enable teams to work together, develop workflows, and move the code to deployment all within the Azure platform. This reduces the overhead and knowledge required to perform DevOps.
- Cultural challenges
 - Teams can often operate in silos if there is no cross-collaboration and this leads to resistance of adoption of more efficient and effective methodology.
 - Azure DevOps cloud environment provides a robust ecosystem that can provide teams with the ability to break down the silos to communicate and collaborate.

intro

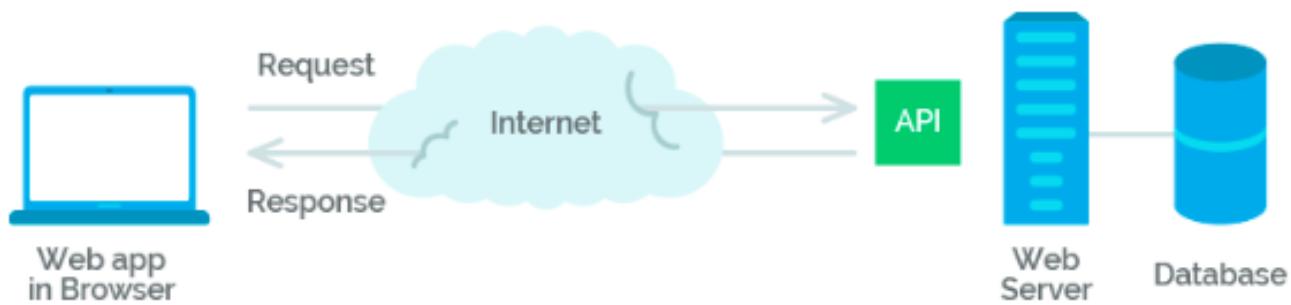
Daily, applications are constantly making requests to databases. Some of those operations are legitimate, while others are spurious and attempting to access data in an unauthorized manner. In this module, topics such as REST services will be covered. REST, or representational state transfer, is a platform for implementing a uniform interface, including constraints, that are used to build web platforms and applications which allow for error handling, database connections, and scalability.

During the course of this module, you will be making use of REST, APIs, SOA, and various testing methodologies and applications. In this module, you will create REST services, SOA, and SOAP services. You will be setting these up in Java and making use of Tomcat as the server to deploy to. Upon completion of this module, you will have a foundation of understanding of REST, SOA, SOAP, and testing of the applications created in this manner.

Restful Services and Applications

Representational State Transfer or REST, is a style for building applications and services. Typically, a REST API, or application programmable interface, is a service built to respond to specific user requests. As we build the applications to respond to specific requests and user needs, our APIs include various URLs where the tools are located to service those requests. These are considered REST endpoints. So when we ask for a new user to be created, we'll call a URL, where the resources needed to carry out this addition of a user reside.

Upon the request to add the user to the system, the URL to do so will not only include the location of the server, but also of a resource or script on the server which will do the actual addition of the user. Depending on what the result of that attempted user addition is, a response will be sent back to the requesting user.



APIs typically fall into the REST or SOAP categories, as both can access web services. SOAP makes use of XML for messaging services while REST makes use of the various HTTP 1.1 verbs to carry out tasks. These verbs are GET, POST, PUT, and DELETE. Our code will make use of various locations to make requests. In this lesson, we will take a look at putting together both REST and SOAP type applications. In addition, we'll be taking a look at the Java REST assured library to test our applications. Finally, you will be working with and using SOA or Service Oriented Architecture to create distributed systems that deliver services to other applications.

REST API

REST has become the defacto standard for building web services because they are easy to build. Also, it is easy to make them usable by many different applications. Consider applications that make use of creating a user account from our earlier example. By making a REST service out of adding a user, we can create a tool that any application can connect to.

REST is built to suit the web and as such, protocols like HTTP provide a host of features that REST can make use of to build responsive and effective applications. REST makes use of verbs like GET, POST, PUT, and DELETE. In addition, REST makes use of encryption and authentication tools in HTTP, along with caching, redirecting, and forwarding.

Springboost

Your xml file will contain the following code:

```
<beans xmlns = "http://www.springframework.org/schema/beans"
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id = "HelloWorld" class = "yourBean.yourBean">
        <property name = "message" value = "Your message"/>
    </bean>

</beans>
```

Add a yourBean.java file. This will have the message output for now, but you will be building more elements into this source file as you go:

```
private String message;

public void setMessage(String message)
{
    this.message = message;
}

public void getMessage()
{
    System.out.println("Your Message : " + message);
}
```

Finally, you will create your StudentDemo.java file, which will contain the main method and will create the xmlApplicationPath and bean creation:

```
// TODO Auto-generated method stub
ApplicationContext context = new ClassPathXmlApplication
Context("beans.xml");
StudentBean obj = (StudentBean) context.getBean("HelloWo
rld");
obj.getMessage();
```

Building out Student Services

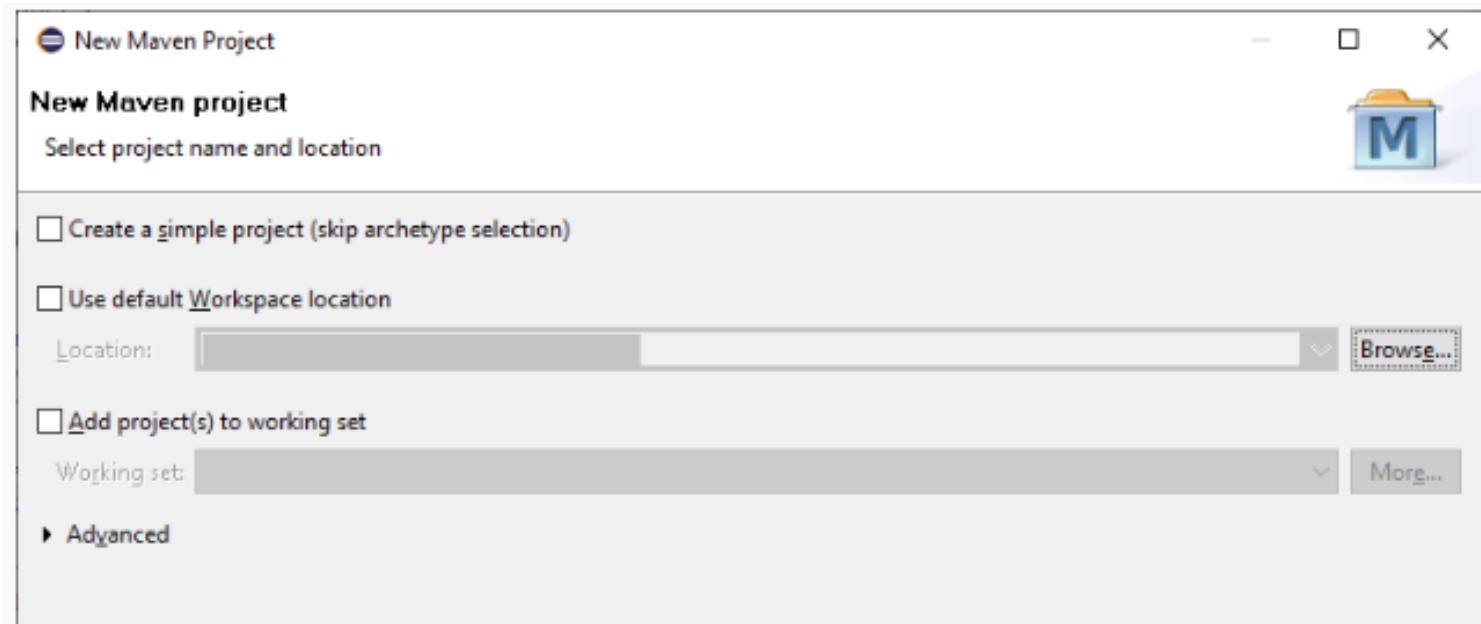
As you continue to build out a set of student services, you will now want to build some browser-based interactions which will allow you to interact with our Students application by passing in a URL with the use of Tomcat.

In this scenario, you are going to pass in some data through the URL string that will allow students to be displayed on the screen. An added scenario to consider would be database support, which will not be covered in this module, but would add more value to the application. Your application will query the service based on data that you submit via the URL, but none of that data will be retained permanently.

In this next lab, you are going to send a student for processing and display that to the screen using a RESTFUL service endpoint. This endpoint will consume the data you send and then display that to the screen. In the hands-on lab, you are then going to add to that service to allow for the removal of those students.

starup.bat

Creating a Maven Project



The screenshot shows the 'Archetype Catalog' dialog. At the top left is a 'Catalog' icon. The title bar says 'All Catalogs'. Below the title, there is a 'Filter' field. The main area is a table with columns 'Group Id', 'Artifact Id', and 'Version'. The table contains the following data:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-quickstart	1.4
org.apache.maven.archetypes	maven-archetype-simple	1.4
org.apache.maven.archetypes	maven-archetype-site	1.4
org.apache.maven.archetypes	maven-archetype-site-simple	1.4
org.apache.maven.archetypes	maven-archetype-site-skin	1.4
org.apache.maven.archetypes	maven-archetype-webapp	1.4
org.apache.myfaces.buildtools	myfaces-archetype-codi-jsf12	1.0.4

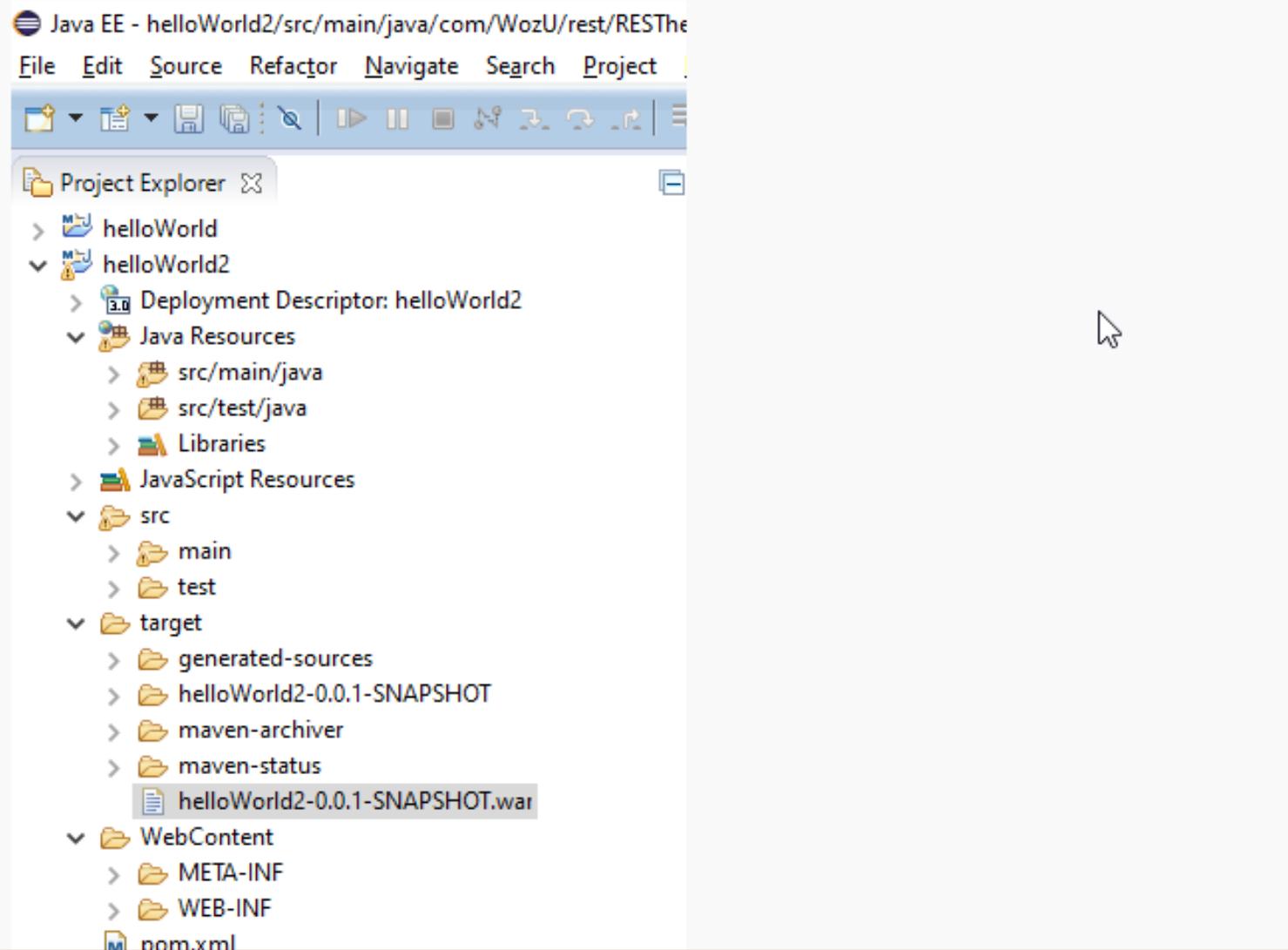
An archetype which contains a sample Maven Webapp project.
<https://repo1.maven.org/maven2>

Checkboxes at the bottom left are 'Show the last version of Archetype only' (checked) and 'Include snapshot archetypes'. A 'Add Archetype...' button is on the right. A 'Advanced' link is at the bottom left. At the bottom are buttons for '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

You can now fill in the Group ID and Artifact ID and click Next. Eclipse will take care of generating the rest of the structure.

Enter Maven Project Details

You will see your application in the project explorer pane:



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.WozU.maven3</groupId>
  <artifactId>helloWorld2</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>helloWorld2 Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <repositories>
    <repository>
      <id>maven2-repository.java.net</id>
      <name>Java.net Repository for Maven</name>
      <url>http://download.java.net/maven/2/</url>
      <layout>default</layout>
    </repository>
  </repositories>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>com.sun.jersey</groupId>
      <artifactId>jersey-server</artifactId>
      <version>1.9</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <sourceDirectory>src/main/java</sourceDirectory>

    <plugins>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.4</version>
        <configuration>
          <warSourceDirectory>WebContent</warSourceDirectory>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
```

```
<version>3.1</version>
<configuration>
  <source>1.8</source>
  <target>1.8</target>
</configuration>
</plugin>

</plugins>

</build>

</project>
```

Web_xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>helloWorld2</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>helloWorld2</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>com.WozU.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>helloWorld2</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>

</web-app>
```

RESTFUL Maven application

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
    <display-name>numbers2</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>numbers2</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>com.WozU.rest</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    .
    <servlet-mapping>
        <servlet-name>numbers22</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

web.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.WozU.maven3</groupId>
  <artifactId>numbers2</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>numbers2 Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <repositories>
    <repository>
      <id>maven2-repository.java.net</id>
      <name>Java.net Repository for Maven</name>
      <url>http://download.java.net/maven/2/</url>
      <layout>default</layout>
    </repository>
  </repositories>
```

```
<dependencies>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-server</artifactId>
    <version>1.9</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
</dependency>
</dependencies>

<build>
<sourceDirectory>src/main/java</sourceDirectory>

<plugins>
```

.....

```
<plugin>
<artifactId>maven-war-plugin</artifactId>
<version>2.4</version>
<configuration>
    <warSourceDirectory>WebContent</warSourceDirectory>
</configuration>
</plugin>

<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<configuration>
    <source>1.8</source>
    <target>1.8</target>
</configuration>
</plugin>

</plugins>

</build>

</project>
```

RestNumber

```
@Path("/numbers")
public class UserRestService {

    // Single Parameter
    @GET
    @Path("{number1}")
    public Response getNumber(@PathParam("number1") int number1) {

        int myNumber = number1 * 2;
        float anotherNumber = number1/2;
        String output = "<h1>Hello World!<h1> - Param 1 is " + number1
+
                "<p>RESTful Service is running ... <br>number2 is " +
myNumber + "</p<br>";

        return Response.status(200).entity("getNumber() method called,
number returned is " + number1 + "]").build();
    }

}
```

Intro to Cucumber and Gherkin

Cucumber and Gherkin

There are several components to any project, IT or otherwise, that are extremely important to ensure that a project is both quality and meets the deliverables specified. One of those elements is QA or quality analysis. In this stage of the lifecycle of a project's development, we are fully vested in putting together testing strategies, scenarios, and methodologies for ensuring that the applications released into the wild are robust, stable, and identifies with the specified deliverables in the project plan.

While there is a need for hands-on testing, there are many facets of testing a project that can be automated. Remember, as we release a project and users begin to interact with it, QA and maintenance will be a constant partner for a project. Users will find bugs and you will release new features, all of which will require further QA to ensure that the patches and updates that are designed into the project are quality and meet with the standards that the organization specifies.

Cucumber

Cucumber is an open-source tool that works with Behavior-Driven Development (BDD). This tool allows anyone to write tests without an in-depth level of knowledge of programming. Cucumber helps to bridge the gap between all manner of organizational personnel to help aid in developing tests, which will ensure a quality application or IT process. Using a combination of Cucumber and Gherkin, users can write scenarios or various types of tests that will describe and test the system's functionality so we can review things from a user's or customer's perspective.

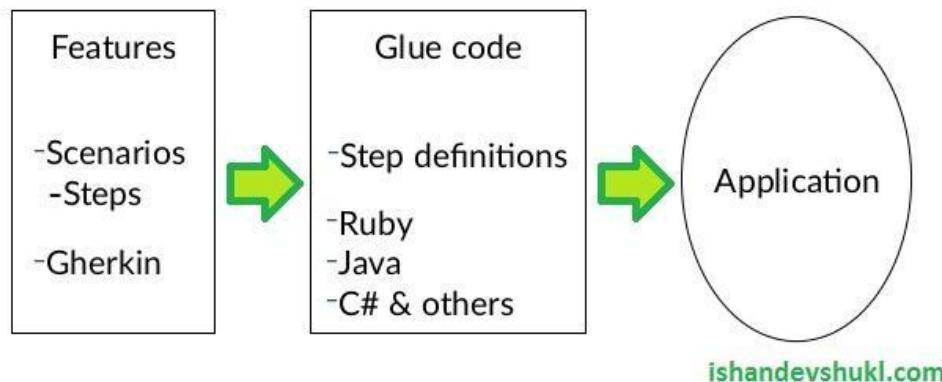
There are a number of scenarios where you can develop tests to ensure that the desired functionality exists and works in an expected and accepted manner. Consider some of the following scenarios, all of which could be used for testing:

1. Testing a login script
2. Filling out forms
3. Searching for resources on a website

Gherkin, simply stated, is the language that uses Cucumber to implement various testing scenarios. Using plain English statements, even non-technical users can help to define tests that will ensure that the business logic and deliverables are built into an application. One of the important measurements of a quality application is whether or not an application meets the business objectives and standards.

Using various keywords, users of any skill type can identify and set up some basic tests. Once those basic set of tests and scenarios are set up, Java code (Glue Code) will need to be written to implement those tests and scenarios:

What is Cucumber?



With your project now setup, you can start to introduce the various components of the Gherkin language to create the desired scenarios to test a web application using Selenium.

As you continue working on building and designing your testing application, including the Cucumber platform, Selenium, and now the Gherkin scenarios, we have a full-fledged testing environment to ensure that the outcomes and objectives that are specified in your project plan are met.

Gherkin allows non-developers to devise and to help to implement testing the various business objectives that are desired for any application. Using basic English statements, users can define the scenarios and elements of an application that are necessary to certify that the application has met certain testing quality standards and that the required business logic has been included in the application.

A user wants to load the website

```
@Given("^Open Chrome and navigate to a site$")
public void open_chrome_and_navigate_to_site() throws Throwable
{
    System.setProperty("webdriver.chrome.driver", "C:\\\\chromedriver\\\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.get("http://www.thecranberryeagle.com/");
    System.out.println("This Step opens Chrome and navigates to a site");
}
```

When: The user enters, email and first name

```
public void enter_the_Email_and_First_Name() throws Throwable
{
    WebElement other = driver.findElement(By.xpath("//input[@data-id='Email Address:input']"));
    other.sendKeys("test@test.com");
    System.out.println("This step enters the Email and First Name fields on the homepage.");
}
```

When: The user enters, email and first name

```
public void enter_the_Email_and_First_Name() throws Throwable
{
    WebElement other = driver.findElement(By.xpath("//input[@data-id='Email Address:input']"));
    other.sendKeys("test@test.com");
    System.out.println("This step enters the Email and First Name fields on the homepage.");
}
```

Then: User clicks on submit and ensures that they are now on the mailing list

```
@Then("^Scrolls down the page and clicks on the link at the bottom$")
public void Scroll_Down_The_Page() throws Throwable
{
    driver.findElement(By.linkText("Contacts")).click();

    JavascriptExecutor js = (JavascriptExecutor)driver;
    driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
    js.executeScript("window.scrollTo(0,500);");

    System.out.println("This section will scroll down the page and .");
}
```

Selenium WebDriver, Cucumber and Gherkin To Interact with a Web Page

Create a folder and a class

```
import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(features="Features", glue={"StepDefinition"})

public class Runner {
```

}

```
public class Steps
{
    private static WebDriver driver = null;

    @Given("^Open Chrome and navigate to a site$")
    public void open_chrome_and_navigate_to_site() throws Throwable
    {
        System.setProperty("webdriver.chrome.driver", "C:\\\\chromeDriver\\\\chromedriver.exe");

        System.out.println("This Step opens Chrome and navigates to a site");
    }

    @When("^Enter the email address and first name$")
    public void enter_the_Email_and_First_Name() throws Throwable
    {

        System.out.println("This step enters the Email and First Name fields on the homepage.");
    }

    @Then("^Scrolls down the page and clicks on the link at the bottom$")
    public void Scroll_Down_The_Page() throws Throwable
    {
        JavascriptExecutor js = (JavascriptExecutor)driver;
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        System.out.println("This section will scroll down the page and .");
    }
}
```

Remember, in each of the keywords for Given, When, and Then, you define the methods which will perform certain actions. This is the GlueCode. In each method, you're going to have to add steps which will perform the actions.

In the Given step, you will navigate to the site:

```
driver = new ChromeDriver();
driver.get("http://www.thecranberryeagle.com/");
In the When step, you will add:
WebElement other = driver.findElement(By.xpath("//input[@data-id='Email Address:input']"));
other.sendKeys("test@test.com");
```

What would you have to do to fill in the first name?

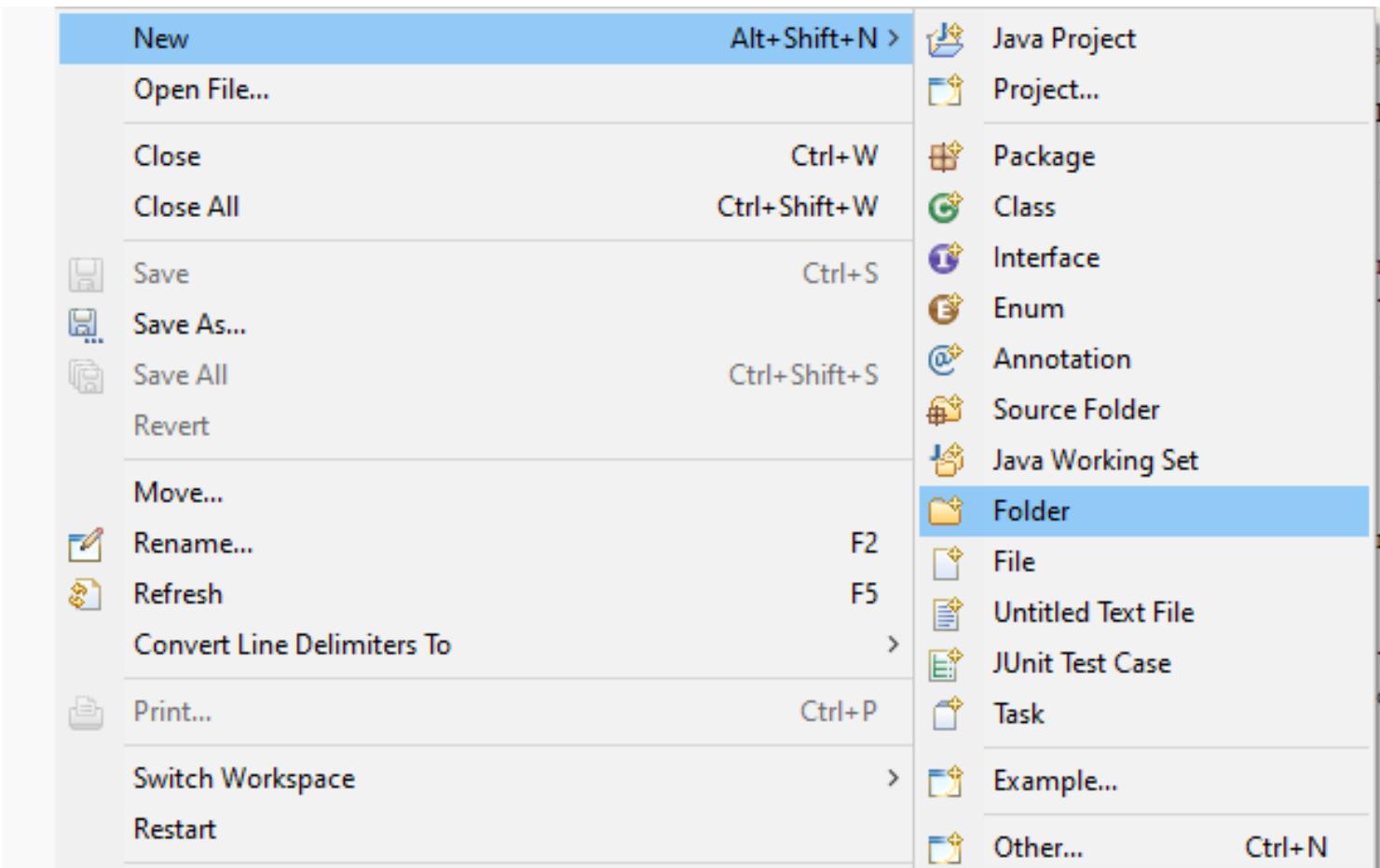
Finally, you will add in the Then step which will scroll down the page and click on the sign-up button. Remember to change the click element to the proper element for that page:

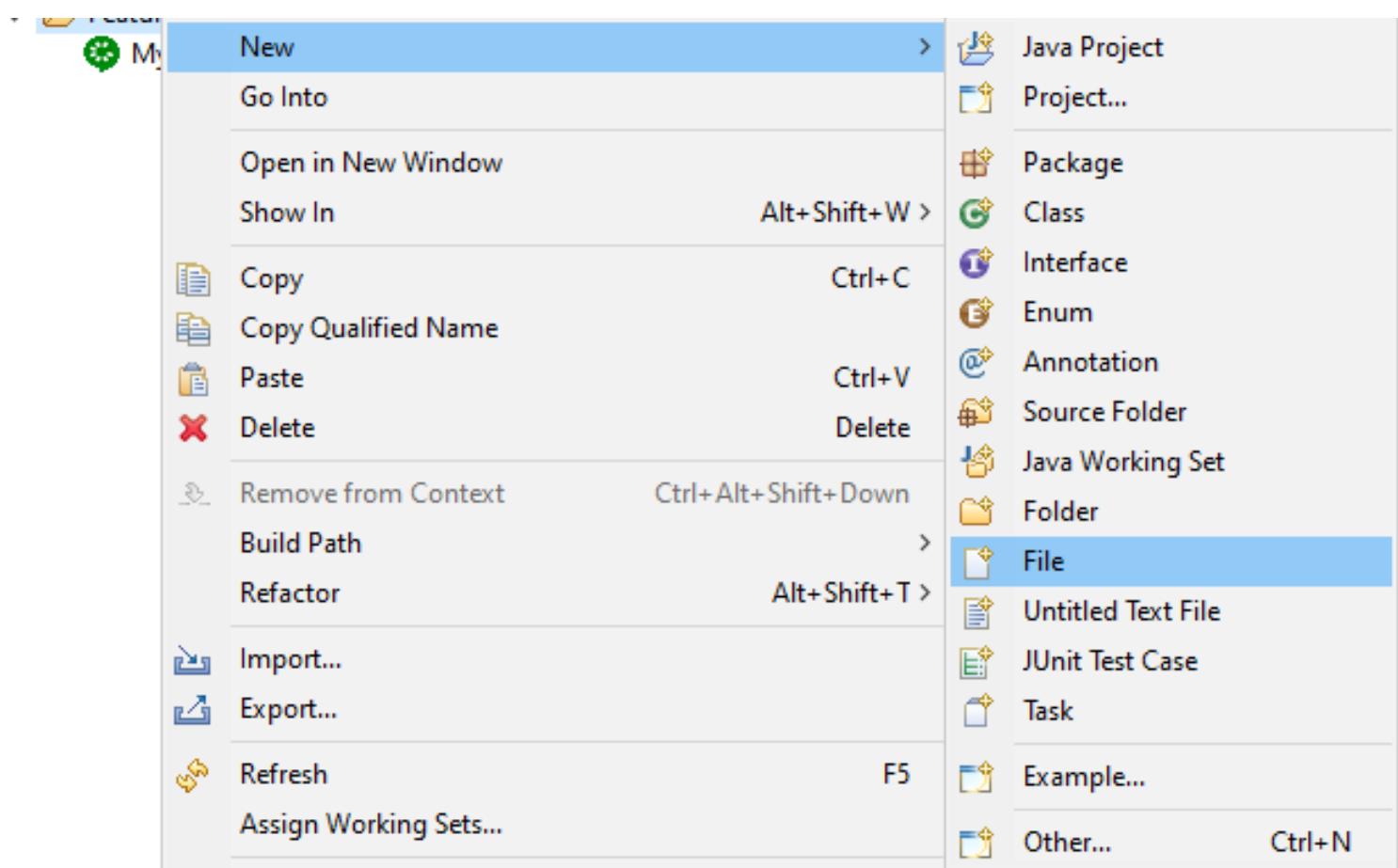
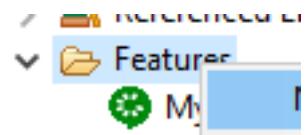
```
JavascriptExecutor js = (JavascriptExecutor)driver;
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
js.executeScript("window.scrollTo(0,500);");

driver.findElement(By.linkText("Contacts")).click();
```

```
JavascriptExecutor js = (JavascriptExecutor)driver;
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
js.executeScript("window.scrollTo(0,1500);");
```

create a scenario that is implemented





```
#Author: your.email@your.domain.com
#Keywords Summary :
#Feature: List of scenarios.
#Scenario: Business rule through list of steps with arguments.
#Given: Some precondition step
#When: Some key actions
#Then: To observe outcomes or validation
#And,But: To enumerate more Given,When,Then steps
#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
#Examples: Container for s table
#Background: List of steps run before each of the scenarios
#""" (Doc Strings)
#| (Data Tables)
#@ (Tags/Labels):To group Scenarios
#<> (placeholder)
#"""
## (Comments)
#Sample Feature Definition Template
@tag
Feature: Testing Selenium and Cucumber Setup
    You will use this to test out various scenarios and features of Selenium
```

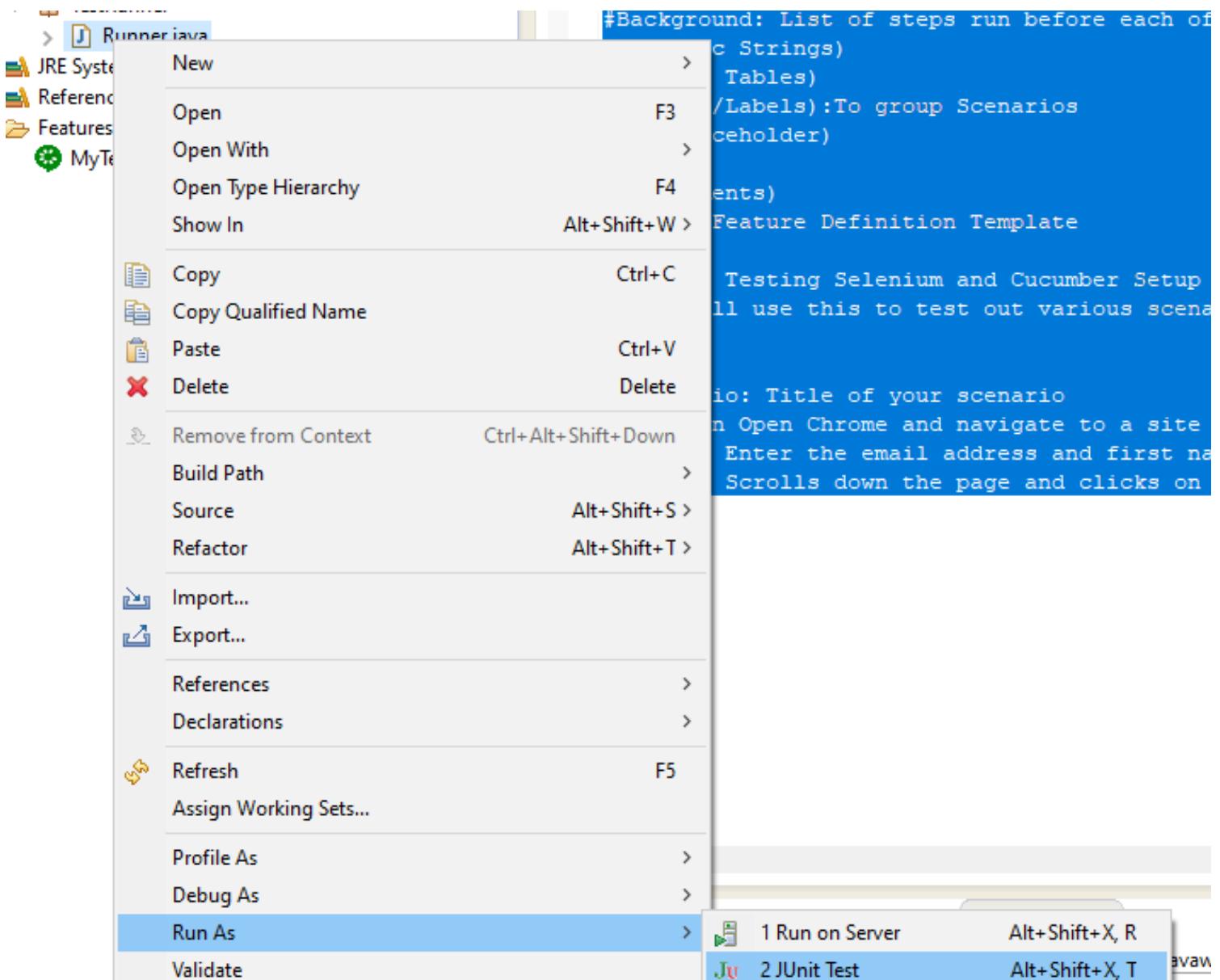
@tag1

Scenario: Title of your scenario

Given Open Chrome and navigate to a site

When Enter the email address and first name

Then Scrolls down the page and clicks on the link at the bottom



DateTable

As you become more comfortable with the Gherkin and Cucumber environments, the ability to customize and make your testing of applications more flexible, you can start to define numerous tests that include various types of data. Web applications center quite a bit around the use of forms and knowing the types of data that are expected in various form fields and then generating a table to define the test of scenarios which will include both good and bad data is important to ensure the stability and that your applications check for bad data.

In this section, you will take a look at how to set up data tables in your feature file so that you can create a number of iterations of tests to include various types of data.

feature file and read the table

```
public void enter_the_Email_and_First_Name(DataTable dt) throws Throwable
{
    List<Map<String, String>> userList = dt.asMaps(String.class, String.class);

    for (int i = 0; i < userList.size(); i++)
    {
        driver.manage().deleteAllCookies();
        WebElement fName = driver.findElement(By.xpath("//input[@name='FNAME']"));
        driver.findElement(By.xpath("//input[@name='FNAME']")).clear();
        fName.sendKeys(userList.get(i).get("fName"));

        WebElement lName = driver.findElement(By.xpath("//input[@name='LNAME']"));
        driver.findElement(By.xpath("//input[@name='LNAME']")).clear();
        lName.sendKeys(userList.get(i).get("lName"));

        WebElement uEmail = driver.findElement(By.xpath("//input[@name='EMAIL']"));
        driver.findElement(By.xpath("//input[@name='EMAIL']")).clear();
        uEmail.sendKeys(userList.get(i).get("userEmail"));

        driver.findElement(By.cssSelector("input[type='submit'][value='Subscribe']")).click();
        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

        System.out.println("Form Submitted");
    }
}
```


FindNumber

```
public static String findNumber(List<Integer> arr, int k) {
    String ans = "NO";
    for(int i : arr){
        if(i == k){
            ans = "YES";
            break;
        }
    }
    return ans;
}
```

OddNumber

```
public static List<Integer> oddNumbers(int l, int r) {
    List<Integer> output = new ArrayList<Integer>();
    if(l % 2 == 0){
        l = l + 1;
    }
    while(l <= r){
        output.add(l);
        l = l + 2;
    }
    return output;
}
```

Question load error

There was a problem with loading this question page.

Server error: Unauthorized request. Looks like you're not logged in (or logged in from elsewhere).

Click continue to goto test listing page.

 **@here** To verify that the counter increases: `@Test public void setUp() { Counter c = new Counter(); int currCount = c.getCount(); c.up(); assertEquals(currCount + 1, c.getCount()); c.up(); assertEquals(currCount + 2, c.getCount()); }` To verify that the counter does not increase past 10: `@Test public void setUpLimit() { Counter c = new Counter(); for (int i = 0; i < 100; i++) { c.up(); assertTrue(c.getCount() <= 10); } }`