Java Real Time Examples

learn java with real time examples

Design Pattern-Real Time

This page talks about how to use design patterns in real time in effective way.
As we know already for each and every problem there will be a solution. That solution can be achieved in many ways. A Pattern tells about among all those solution choose the best one and apply for your requirement.

In Java the problems which comes recurrently for the requirements we are dealing we can go for pre-defined pattern to solve that requirement.

Singleton Design Pattern

Singletons are useful only when you need one instance of a class and it is undesirable to have more than one instance of a class.

When designing a system, you usually want to control how an object is used and prevent users (including yourself) from

making copies of it or creating new instances. For example, you can use it to create a connection pool. It’s not wise to create a new connection every time a program needs to write something to a database; instead, a connection or a set of connections that are already a pool can be instantiated using the Singleton pattern.

Example - Property file reading.

Usually we deal with the property files , property files can be loaded once, and though out the application we can use the object without re instance creation. In this scenario we can make use of Singleton Design Pattern

Here is the code to make use of singleton pattern by reading property file

```
package com;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;
import java.lang.System;

public class SingletonDesign{
 /**
  * @param args
  */
 public static void main(String[] args) {
  // TODO Auto-generated method stub
  SingleTon ton = SingleTon.getSingleTonObject();
  System.out.println(ton.getProperties().get("name"));
  System.out.println(ton.getProperties().get("blogname"));

  ton.getProperties().setProperty("email", "something@gmail.com");
  System.out.println(ton.getProperties().get("email"));
```

```
 }


}
class SingleTon{
 public static SingleTon singleTon=null;
 public Properties properties=null;

 public Properties getProperties() {
  return properties;
 }
 public void setProperties(Properties properties) {
  this.properties = properties;
 }
 private SingleTon(){

 }
 public static synchronized SingleTon getSingleTonObject(){
  if(singleTon==null){
   singleTon = new SingleTon();
   Properties properties1 = new Properties();
   try{
    properties1.load(new FileInputStream("D:/propertiesFile.properties"));
    singleTon.setProperties(properties1);
   }catch(FileNotFoundException ex){
    ex.printStackTrace();
   }catch(IOException ioe){
    ioe.printStackTrace();
   }
   return singleTon;
  }
  return singleTon;
 }
 public Object clone() throws CloneNotSupportedException{
  throw new CloneNotSupportedException();
 }

}
```


About Me

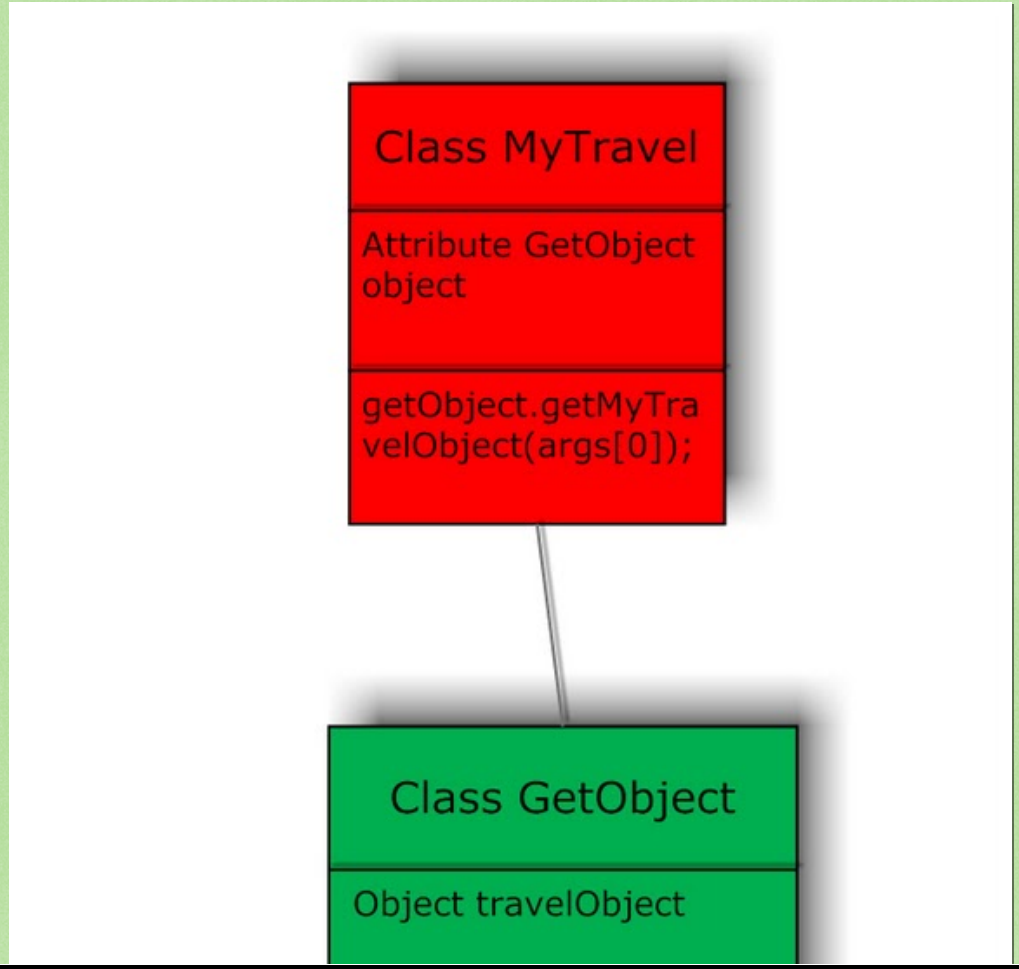
suneel

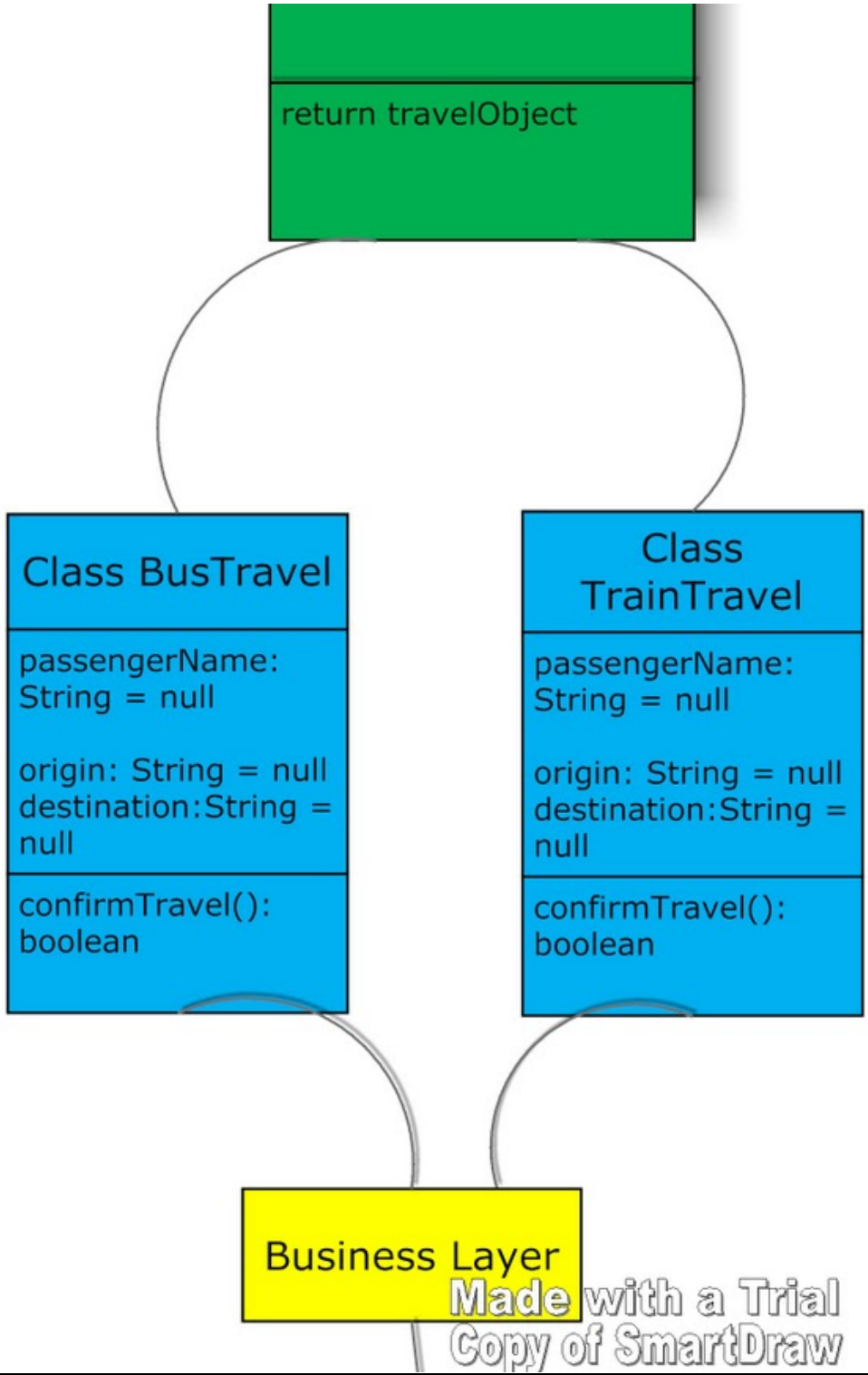
View my complete profile

outcome
---
suneel
http://suneel-javautils.blogspot.com
something@gmail.com

Factory Design Pattern

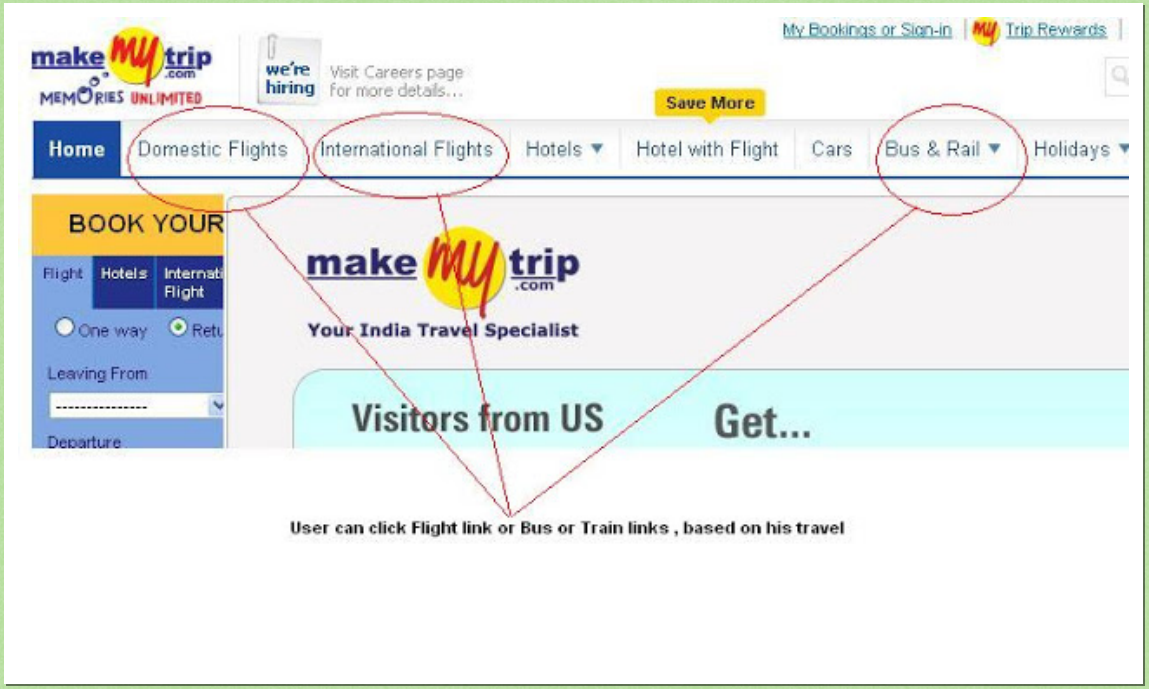
Factory Design pattern explains about how get the desired object from the factory based on the requirement.
here is the picture representation of Factory Design Pattern





In the above flow diagram i have explained how to get the desired object based on the input (bus or train) user has provided.

for ex: **www.makemytrip.com** is the official site , where that site allow us to book train ticket as well bus tickets and flight ticket also. In this case user can click either bus ticket link or train ticket link or flight ticket link.



we can not make the static binding with one class with one Travel Object , So there should be a intermediate class which takes travel type parameter then according to the parameter it will instantiate the desired object.

The classes which i have used

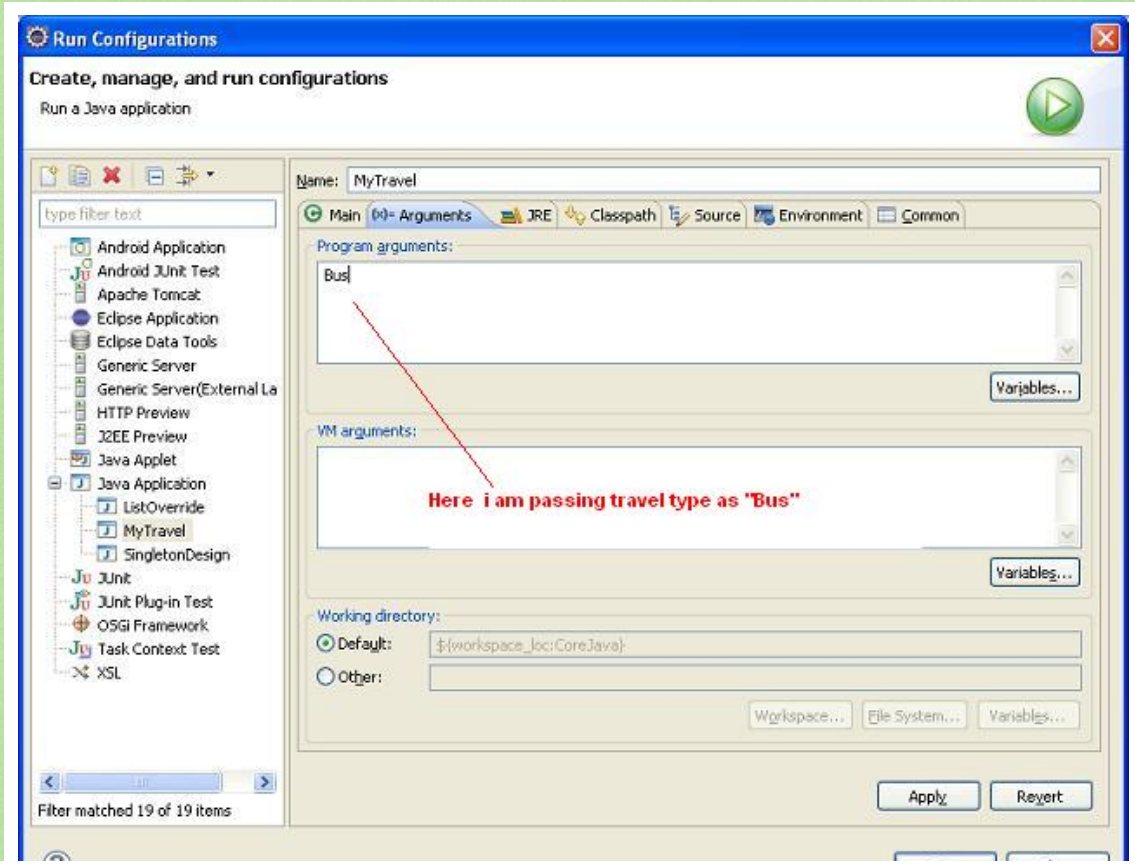
MyTravel : is the client class which take the travel type like : Bus or Train or Flight
GetObject : is a generic class which will instantiate the Bus or Train and it returns the generic object.
BusTravel : is POJO with the passenger name, Origin, destination parameters
TrainTravel : is the POJO with the passenger name, Origin , destination parameters.

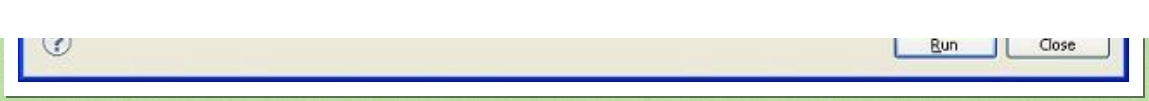
```
package com;


class BusTravel{
 String passengerName;
 String origin;
 String destination;

 public boolean confirmTravel(){
  if(this.passengerName!=null && this.origin!=null && this.destination!=null){
   //DB query here to check seats availability and if it return true
   System.out.println("Passenger choose Bus Travel");
   return true;
  }else{
   //DB query here to check seats availability and if it return false
   return false;
  }
```

```
 }
 public String getPassengerName() {
  return passengerName;
 }
 public void setPassengerName(String passengerName) {
  this.passengerName = passengerName;
 }
 public String getDestination() {
  return destination;
 }
 public void setDestination(String destination) {
  this.destination = destination;
 }
 public String getOrigin() {
  return origin;
 }
 public void setOrigin(String origin) {
  this.origin = origin;
 }
 public String toString(){
  return "BusObject";
 }
}
class TrainTravel {
 String passengerName;
 String origin;
 String destination;

 public boolean confirmTravel(){
  if(this.passengerName!=null && this.origin!=null && this.destination!=null){
   //DB query here to check seats availability and if it return true
   System.out.println("Passenger choose Train Travel");
   return true;
  }else{
   //DB query here to check seats availability and if it return false
   return false;
  }
 }
 public String getPassengerName() {
```

```
  return passengerName;
 }
 public void setPassengerName(String passengerName) {
  this.passengerName = passengerName;
 }
 public String getDestination() {
  return destination;
 }
 public void setDestination(String destination) {
  this.destination = destination;
 }
 public String getOrigin() {
  return origin;
 }
 public void setOrigin(String origin) {
  this.origin = origin;
 }
 public String toString(){
  return "TrainObject";
 }


}
class GetObject{
 public Object getMyTravelObject(String travelType){
  Object travelObject = null;
  if("Bus".equalsIgnoreCase(travelType)){
   System.out.println("travel type:"+travelType);
   travelObject = new BusTravel();
  }else if("Train".equalsIgnoreCase(travelType)){
   System.out.println("travel type:"+travelType);
   travelObject = new TrainTravel();
  }
  return travelObject;
 }
}
public class MyTravel{
 public static void main(String[] args) {
  // TODO Auto-generated method stub
```

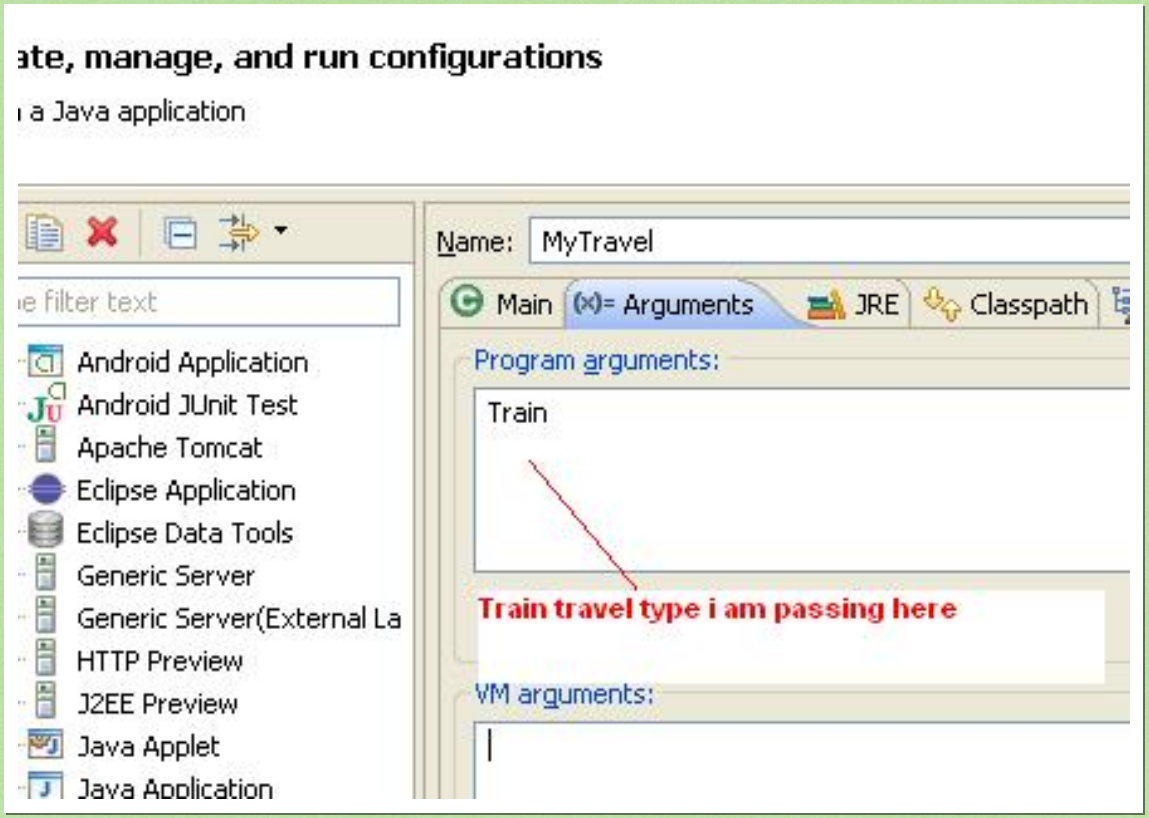
```
 GetObject getObject = new GetObject();
 Object travelType = getObject.getMyTravelObject(args[0]);
 System.out.println("Travel request type :"+args[0]);


 if(travelType instanceof BusTravel){
  System.out.println("Yes Bus object instatiated");
 }else if(travelType instanceof TrainTravel){
  System.out.println("Yes Train object instatiated");
 }else{
  System.out.println("Pass the travel parameter correctly");
 }
}


}
```





outcome
---

travel type:Bus
Travel request type :Bus
Yes Bus object instatiated



outcome

---

travel type:Train
Travel request type :Train
Yes Train object instantiated

If the travel does not match with neither "Bus" nor "Train" then the out come will be

Travel request type :Something
Pass the travel parameter correctly

that's all :)

Facade Design Pattern

Observer Design Pattern

MVC Design Pattern

Business Delegate

Data Access Object Design Pattern

Front Controller Design Pattern

Service Locator

Transfer Object or Value Object

Home

Subscribe to: Posts (Atom)

Watermark template. Powered by Blogger.