# 项目计划书

**项目名称：火车订票系统**

## 引言

### 1.1 编写目的

为了锻炼我们的软件编程能力以及团队软件开发的协作水平，并保证团队能够按时完成课题任务，务必要把任务流程、成员的工作职责、软件的过程模型写入文件中保存下来。

### 1.2 项目背景

本项目是一款火车订票系统，订票系统包括浏览当前所有火车的信息，包含起始站与终点站，途径的站点，发车日期和时间，到站时刻、余票情况、座位和票价；查询功能包含对车次的查询，也可以按照区间进行查询，其次是对已经购买订单的查询，线下也可以进行购票，数据库中的余票情况相应减少，购买功能包含选定日期、车次和区间后进行购买，同时线下购票时也会相应减少数据库中的车票记录，目的是方便人们日常购买车票。

### 1.3 项目开发时间

项目启动时间：2021-3-17 项目预计上线时间：2021-6-15

### 1.4 参考资料

[1]王红霞. Java 程序设计基础教程[M].北京：中国电力出版社，2009.07.

[2]陆凌牛. HTML5与CSS3权威指南(第3版 上册)[M]. 机械工业出版社,2015.09.

[3]商恩福,网上订票系统分析与设计[J].电脑编程技巧与维护,2009(16):13-14

[4]刘立新,田磊,王文强. 基于SMS的火车票票务管理系统[J]. 西安邮电大学学报,2008,13(3):62-66.

## 项目概述

### 2.1 团队架构及分工

本小组具体成员：李苗、霍达、文春林、赵德曹

小组成员分工如下：

霍达：项目经理、软件测试

李苗：需求分析师、文档编写人员

文春林：软件开发工程师

赵德曹：软件测试、数据库设计师

### 2.2 项目开发流程

**（1）3月17日-3月27日**

整个系统的构思和流程，包括使用人员的种类、系统要实现的主要功能

**（2）3月28日-4月28日**

前端页面设计和编码，包括用户登陆页面、用户注册界面、用户个人信息页面、购票首页、火车查询页面、火车票购买页面、各种管理页面。

**（3）4月29日-5月29日**

后端功能的实现、数据库的设计和连接

**（4）5月30日-6月15日**

系统测试，完成用户使用手册，将系统投入试运营。

### 2.3 软件开发模型

我们组使用的软件开发模型是瀑布模型，其特点是：

（1）阶段间具有顺序性和依赖性：必须等前一个阶段的工作完成后，才能开始下一个阶段的工作；前一阶段的输出文档是下一个阶段的输入文档。

（2）推迟实现的观点：清楚地区分逻辑设计和物理设计，尽可能推迟程序的物理实现。

（3）质量保证观点：每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务；每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

瀑布模型是一种文档驱动模型。

优点：

（1）可以保证整个软件产品较高的质量,保证缺陷能够提前的被发现和解决；

（2）保证系统在整体上的充分把握,使系统具备良好的扩展性和可维护性.

缺点：

（1）缺乏灵活性，当客户对软件需求不明确，或者需求经常变化的情况下，可能导致最终开发出来的软件不能真正满足客户需要。

（2）对于中小型的项目，需求设计和开发人员往往在项目开始后就会全部投入到项目中，而不是分阶段投入，因此采用瀑布模型会导致项目人力资源过多的闲置的情况，这也是必须要考虑的问题。

采用原因：对于订票系统，一般无论是购买车票还是车票的管理员需求比较明确，基本上一次性可以确定全部所需要的需求，因此这个项目比较适合瀑布模型。

### 2.4 系统规模估算

1.LOC

|  |  |
| --- | --- |
| Functions | Estimated LOC |
| 管理员功能 | 108 |
| 公司功能 | 295 |
| 购票功能 | 1092 |
| 订单功能 | 547 |
| 用户功能 | 112 |
| 验证功能 | 357 |

计算过程：

管理员功能 (50+4\*100+200)/6=108

公司功能 (120+4\*300+450)/6=295

购票功能 (850+4\*1100+1300)/6=1092

订单功能 (450+4\*520+750)/6=547

用户功能 (70+4\*100+200)/6=112

验证功能 (200+4\*360+500)/6=357

2.成本

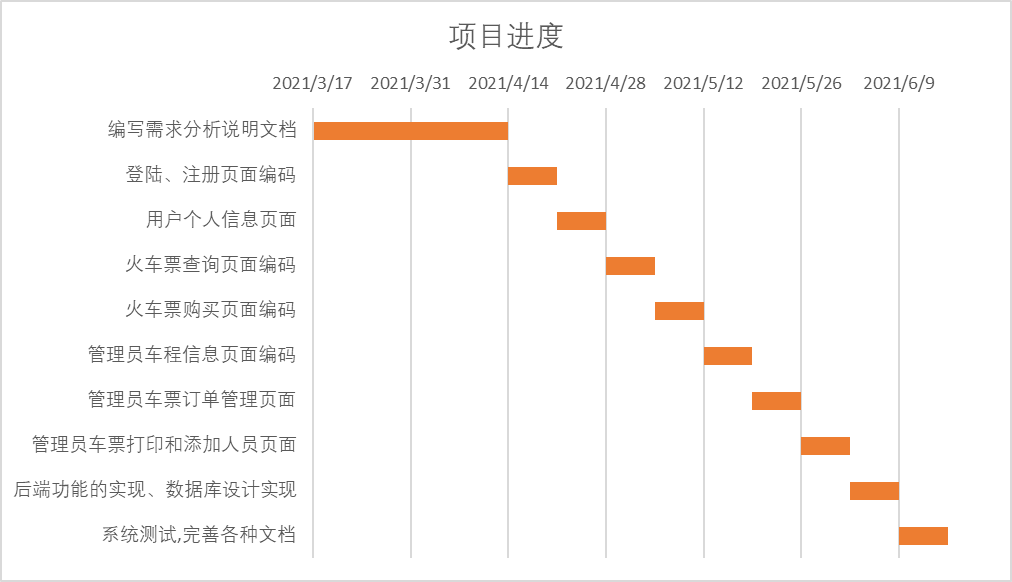
四个人 项目工期三个月 每个人工作日工资400元

工资 400\*（30-8）\*3\*4=105600元

### 2.5 项目进度计划

|  |  |  |  |
| --- | --- | --- | --- |
| 项目进度计划表 | | |  |
| 序号 | 项目进度 | 开始时间 | 周期 |
| 1 | 编写需求分析说明文档 | 2021/3/17 | 28 |
| 2 | 登陆、注册页面编码 | 2021/4/14 | 7 |
| 3 | 用户个人信息页面 | 2021/4/21 | 7 |
| 4 | 火车票查询页面编码 | 2021/4/28 | 7 |
| 5 | 火车票购买页面编码 | 2021/5/5 | 7 |
| 6 | 管理员车程信息页面编码 | 2021/5/12 | 7 |
| 7 | 管理员车票订单管理页面 | 2021/5/19 | 7 |
| 8 | 管理员车票打印和添加人员页面 | 2021/5/26 | 7 |
| 9 | 后端功能的实现、数据库设计实现 | 2021/6/2 | 7 |
| 10 | 系统测试,完善各种文档 | 2021/6/9 | 7 |

项目进度甘特图



## 编码规范

本Java代码规范以SUN的标准Java代码规范为基础，本文档中没有说明的地方，请参看SUN Java标准代码规范。如果两边有冲突，以SUN Java标准为准。

### 1. 标识符命名规范

#### 1.1 概述

标识符的命名力求做到统一、达意和简洁。

##### 1.1.1 统一

统一是指，对于同一个概念，在程序中用同一种表示方法，比如对于供应商，既可以用supplier，也可以用provider，但是我们只能选定一个使用，至少在一个Java项目中保持统一。统一是作为重要的，如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。

##### 1.1.2 达意

达意是指，标识符能准确的表达出它所代表的意义，比如： newSupplier, OrderPaymentGatewayService等；而 supplier1, service2，idtts等则不是好的命名方式。准确有两成含义，一是正确，二是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier1, 远没有targetSupplier意义丰富。

##### 1.1.3 简洁

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWhichIsTransfered 太长， transferedTargetSupplierOrderName则较好，但是transTgtSplOrdNm就不好了。省略元音的缩写方式不要使用，我们的英语往往还没有好到看得懂奇怪的缩写。

##### 1.1.4 骆驼法则

Java中，除了包名，静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过首字母大写来分割。比如: supplierName, addNewContract，而不是 supplier\_name, add\_new\_contract。

##### 1.1.5 英文 vs 拼音

尽量使用通俗易懂的英文单词，如果不会可以向队友求助，实在不行则使用汉语拼音，避免拼音与英文混用。比如表示归档，用archive比较好, 用pigeonhole则不好，用guiDang尚可接受。

#### 1.2 包名

使用小写字母如 com.xxx.settlment，不要 com.xxx.Settlement

单词间不要用字符隔开，比如 com.xxx.settlment.jsfutil，而不要com.xxx.settlement.jsf\_util

#### 1.3 类名

##### 1.3.1 首字母大写

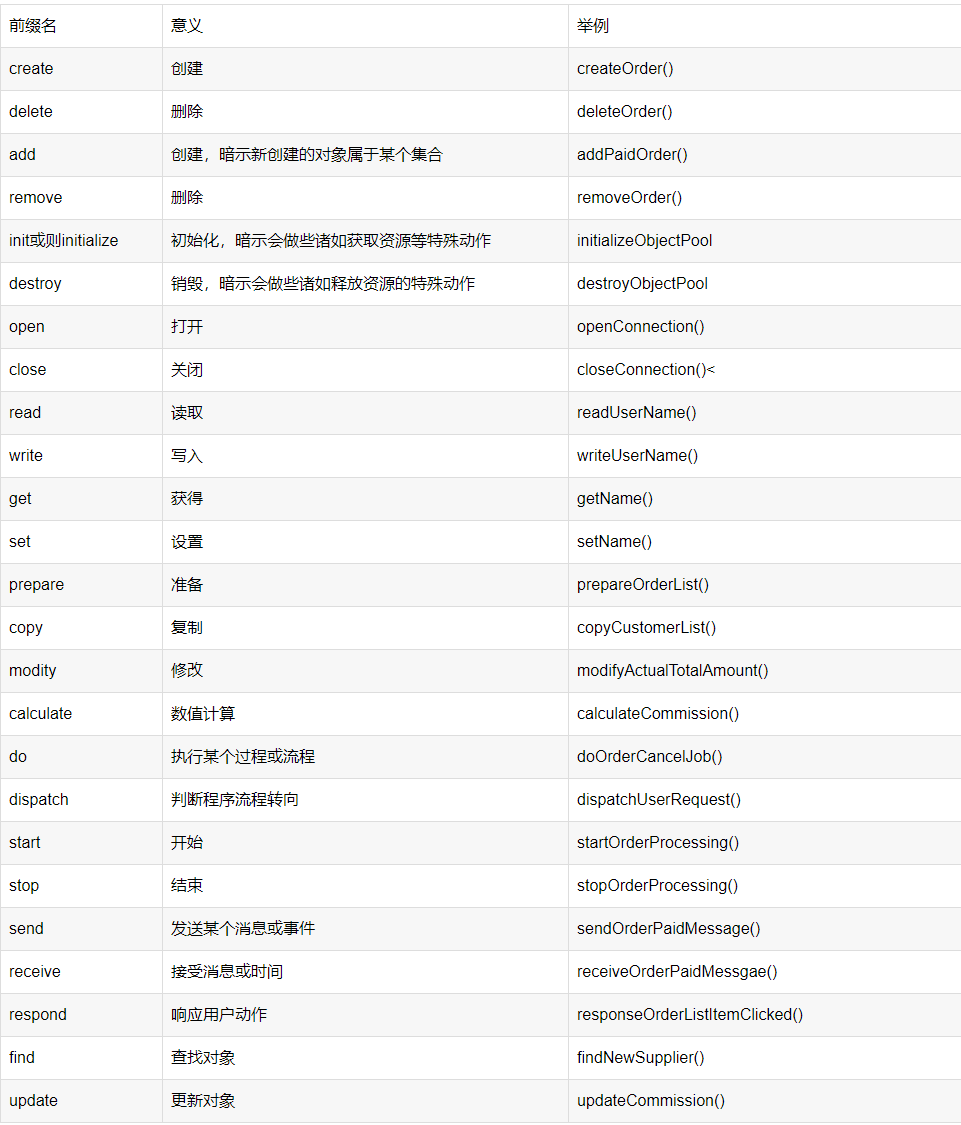
类名要首字母大写，比如 SupplierService, PaymentOrderAction；不要 supplierService, paymentOrderAction.

##### 1.3.2 后缀

类名往往用不同的后缀表达额外的意思，如下表：



#### 1.4 方法名



find方法在业务层尽量表达业务含义，比如 findUnsettledOrders()，查询未结算订单，而不要findOrdersByStatus()。 数据访问层，find,update等方法可以表达要执行的sql，比如findByStatusAndSupplierIdOrderByName(Status.PAID, 345)

#### 1.5 域（field）名

##### 1.5.1 静态常量

全大写用下划线分割，如

public static find String ORDER\_PAID\_EVENT = “ORDER\_PAID\_EVENT”;

##### 1.5.2 枚举

全大写，用下划线分割，如

public enum Events {

ORDER\_PAID,

ORDER\_CREATED

}

##### 1.5.3 其他

首字母小写，骆驼法则，如：

public String orderName;

#### 1.6 局部变量名

参数和局部变量名首字母小写，骆驼法则。尽量不要和域冲突，尽量表达这个变量在方法中的意义。

### 2. 代码格式

用空格字符缩进源代码，不要用tab，每个缩进4个空格。

#### 2.1 源文件编码

源文件使用utf-8编码，结尾用unix n 分格。

#### 2.2 行宽

行宽度不要超过80。Eclipse标准

#### 2.3 包的导入

删除不用的导入，尽量不要使用整个包的导入。在eclipse下经常使用快捷键 ctrl+shift+o 修正导入。

#### 2.4 类格式

#### 2.5 域格式

每行只能声明一个域。

域的声明用空行隔开。

#### 2.6 代码块格式

##### 2.6.1 缩进风格

大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首，例如：

package com.test;

public class TestStyle extends SomeClass implements AppleInter, BananaInter {

    public static final String THIS\_IS\_CONST = "CONST VALUE";

    private static void main(String[] args) {

        int localVariable = 0;

    }

    public void compute(String arg) {

        if (arg.length() >  0) {

            System.out.println(arg);

        }

        for (int i = 0; i < 10; i++) {

              System.out.println(arg);

        }

       while (condition) {

       }

      do {

          otherMethod();

      } while (condition);

  switch (i) {

  case 0:

     callFunction();

     break;

  case 1:

     callFunctionb();

     break;

  default:

     break;

  }

 }

}

##### 2.6.2 空格的使用

###### 2.6.2.1 表示分割时用一个空格

不能这样：

if       (               a >        b   )            {

    //do something here

};

###### 2.6.2.2 二元三元运算符两边用一个空格隔开

如下：

a + b = c;

b - d = e;

return a == b ? 1 : 0;

不能如下：

a+b=c;

b-d=e;

return a==b?1:0;

###### 2.6.2.3 逗号语句后如不换行，紧跟一个空格

如下：

call(a, b, c);

不能如下：

call(a,b,c);

##### 2.6.3 空行的使用

空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码， 如下：

order = orderDao.findOrderById(id);

//update properties

order.setUserName(userName);

order.setPrice(456);

order.setStatus(PAID);

orderService.updateTotalAmount(order);

session.saveOrUpdate(order);

上例中的空行，使注释的作用域很明显.

连续两行的空行代表更大的语义分割。

方法之间用空行分割

域之间用空行分割

超过十行的代码如果还不用空行分割，就会增加阅读困难

### 3. 注释规范

#### 3.1 注释 vs 代码

注释宜少而精，不宜多而滥，更不能误导

命名达意，结构清晰， 类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂；相反，代码混乱，再多的注释都不能弥补。所以，应当先在代码本身下功夫。

不能正确表达代码意义的注释，只会损害代码的可读性。

过于详细的注释，对显而易见的代码添加的注释，罗嗦的注释，还不如不写。

注释要和代码同步，过多的注释会成为开发的负担

注释不是用来管理代码版本的，如果有代码不要了，直接删除，svn会有记录的，不要注释掉，否则以后没人知道那段注释掉的代码该不该删除。

#### 3.2 Java Doc

表明类、域和方法等的意义和用法等的注释，要以javadoc的方式来写。Java Doc是给类的使用者来看的，主要介绍 是什么，怎么用等信息。凡是类的使用者需要知道，都要用Java Doc 来写。非Java Doc的注释，往往是个代码的维护者看的，着重告述读者为什么这样写，如何修改，注意什么问题等。 如下：

/\*\*

\* This is a class comment

\*/

public class TestClass {

    /\*\*

    \* This is a field comment

    \*/

    public String name;

    /\*\*

    \* This is a method comment

    \*/

    public void call() {

    }

}

#### 3.3 块级别注释

3.3.1 块级别注释，单行时用 //, 多行时用 /\* .. \*/。

3.3.2 较短的代码块用空行表示注释作用域

3.3.3 较长的代码块要用

/\*------ start: ------\*/

和

/\*-------- end: -------\*/

包围

如：

/\*----------start: 订单处理 ------- \*/

//取得dao

OrderDao dao = Factory.getDao("OrderDao");

/\* 查询订单 \*/

Order order = dao.findById(456);

//更新订单

order.setUserName("uu");

order.setPassword("pass");

order.setPrice("ddd");

orderDao.save(order);

/\*----------end: 订单处理 ------- \*/

3.3.4 可以考虑使用大括号来表示注释范围

使用大括号表示注释作用范围的例子：

/\*----------订单处理 ------- \*/

{

 //取得dao

 OrderDao dao = Factory.getDao("OrderDao");

 /\* 查询订单 \*/

 Order order = dao.findById(456);

 //更新订单

 order.setUserName("uu");

 order.setPassword("pass");

 order.setPrice("ddd");

 orderDao.save(order);

}

#### 3.4 行内注释

行内注释用 // 写在行尾

## Java代码检查列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **包/类/接口定义检查** | | | | |
| 序号 | 检查项 | | 检查结果 | 备注 |
| 1 | 包/类/接口命名是否合理，是否符合逻辑要求? | |  |  |
| 2 | 类中属性、方法的可访问性是否降低到最小？  （少使用public） | |  |  |
| 3 | 类的代码行数是否超过2000行？ | |  |  |
| 4 | 类中的变量\函数命名是否清晰明确？ | |  |  |
| 5 | 类的接口是否能让人清除明白的知道该如何调用它？ | |  |  |
| 6 | 类是否对其使用者，包括其派生类做了假设？ | |  |  |
| **函数、变量、语句** | | | | |
| 1 | 参数在使用前是否进行了检查？ |  | |  |
| 2 | 函数名称是否有好的内聚性，是否只做一件事，并把它做好？ |  | |  |
| 3 | 是否消除了魔术数字（使用常量代替） |  | |  |
| 4 | 变量名是否明确，是否是单一用途： |  | |  |
| 5 | 变量定义是否准确表达了所代表的含义，是否与其他变量容易记错？ |  | |  |
| 6 | 声明变量定义同时是否进行了初始化，每个变量是否只有一项用途？ |  | |  |
| 7 | 是否有长度超过80字符的代码行？ |  | |  |
| 8 | 是否合理使用的缩进，换行，增加代码可读性 |  | |  |
| 9 | 循环变量的初始值、结束值是否正确，循环的次数是否正确，在循环体内修改循环变量值了吗？ |  | |  |
| **注释** | | | | |
| 1 | 类是否有注释，包括名称、用途，修改记录等 |  | |  |
| 2 | 函数有注释？功能、输入、返回及异常的注释（重要方法）? |  | |  |
| 3 | 复杂的分支流程是否有注释被注释？  距离较远的 } 是否已经被注释？ |  | |  |
| **4** | 注释是否与代码一致 |  | |  |
| **异常处理** | | | | |
| 1 | 代码中是否使用了空的catch语句，是否有 e.printStackTrace()，直接输出错误堆栈 |  | |  |
| 2 | 每个抛出的异常中是否的包含了所有的背景信息 |  | |  |
| 3 | 在catch中捕获的异常是否够？（注意对throwable异常的处理） |  | |  |
| **附加** | | | | |

## 代码检查工具

FindBugs是一个能静态分析源代码中可能会出现Bug的Eclipse插件工具。它也可以分析被编译过的程序。这个工具已经被下载了超过70万次。