成绩

中国农业大学

课程论文

(2020 -2021 学年夏季学期)

论文题目：实训报告

课程名称：农业信息化应用开发实训 I

任课教师：雷宏洲

班　　级：试验 202

学　　号：2019308160102

姓　　名：张语嫣

实训报告

一、基础概念题

（1）观察下图，找出其中所包含对象。

1）用 Java 语言描述这些对象，提供类的基本域和方法？

面向对象分析：

图中所给对象共有太阳,火星，地球，金星，水星，木星，小行星带，土星，天王星，海王星，冥王星 11 个，根据维基百科(zh.wikipedia.org/wiki/太阳系天体列表)，将这些对象分类如下并给它们按百科编了一些属性和方法：

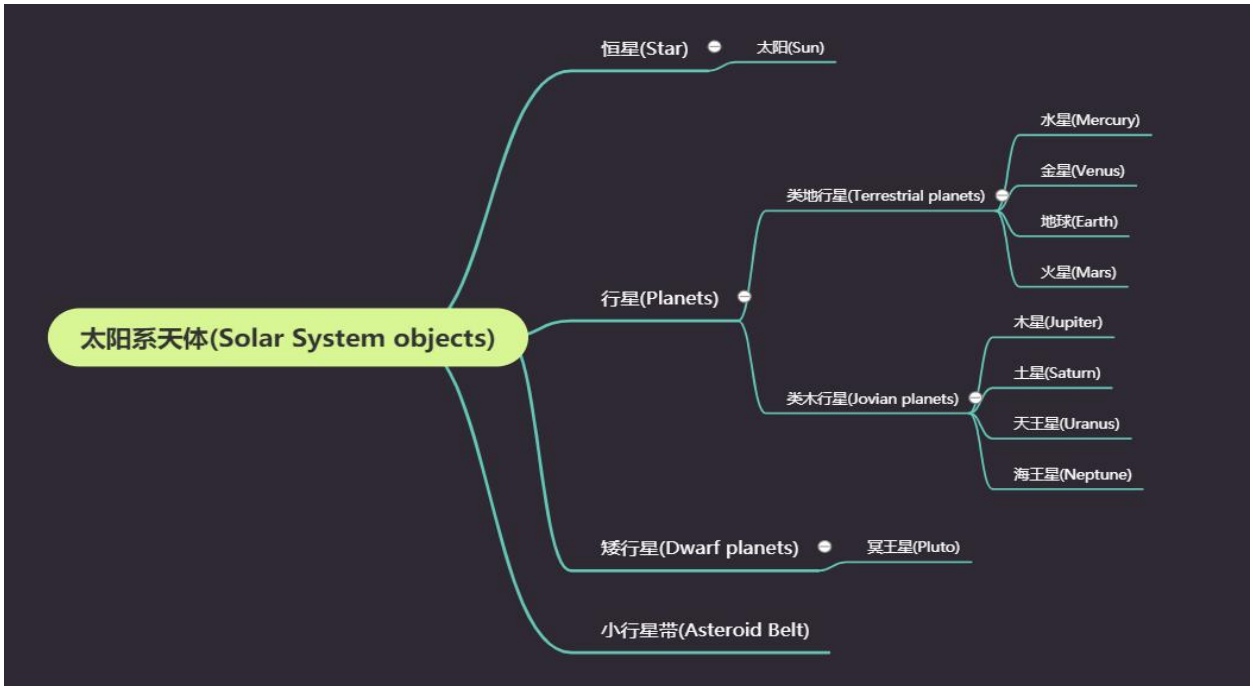


Fig 1 抽象关系

具体属性、方法和继承关系可见 2)图

用 Java 语言描述如下：

```
public abstract class Solar_system_object{
    public String name;
    public void set_name(String name)
    {
        this.name = name;
    }
    public abstract void rotation();
    public abstract void revolution();
}
```

```java
public abstract class Star extends Solar_system_object {
    public double mass;
    public int surface_temperature;
    public double radius;
    public abstract void glowing();
}

public abstract class Planet extends Solar_system_object {
    public double distance;
    public double radius;
    public double mass;
    public boolean moon;
    public void revolution()
    {
        System.out.println("行星环绕太阳公转");
    }
    public abstract void print_relative_location();

}


public abstract class Dwarf_planet extends Solar_system_object {
    public double distance;
    public double radius;
    public double mass;
    public boolean moon;
    public void revolution()
    {
        System.out.println("矮行星沿倾斜轨道环绕太阳公转");
    }
}

public class Asteroid_Belt extends Solar_system_object {
    public int num;
    public String biggest_object;
    public double[] distance;
    Asteroid_Belt() {
        this.set_name("Asteroid Belt");
        this.num = 500000;
        this.distance = new double[] {2.17, 3.64};
        this.biggest_object = "谷神星";
    }

    public void rotation()
    {
        System.out.println("小行星带里的每个小行星都会自转");
```

```
    }
    public void revolution()
    {
        System.out.println("小行星带里的每个小行星都会绕太阳公转");
    }
    public void print_relative_location() {
        System.out.println("介于火星和木星轨道之间，距离太阳"+distance[0]+"天文单位到
"+distance[1]+"天文单位之间");
    }
}

public class Sun extends Star {
    public boolean sunspots;
    public boolean solarflare;
    Sun()
    {
        this.sunspots = true;
        this.set_name("Sun");
        this.radius = 6.955*1e5;
        this.surface_temperature = 6000;
        this.mass = 1.9891*1e30;
    }
    public void rotation()
    {
        System.out.println("太阳自转，其自转周期是 25.05 天");
    }
    public void revolution()
    {
        System.out.println("太阳绕银河系公转，其公转周期约为 2.5×10⁸年");
    }
    public void glowing()
    {
        System.out.println("太阳内部发生核聚变和电磁辐射发光发热，其表面温度可达到
"+surface_temperature+"K");
    }
    public void solar_flare_activity(boolean is_active)
    {
        this.solarflare = is_active;
        if(this.solarflare)
            System.out.println("太阳耀斑正在爆发");
        else
            System.out.println("太阳耀斑没在爆发");
    }
}
```

```
public abstract class Terrestrial_planet  extends Planet{
    public String component = "硅酸盐";
    public boolean ishabitable;
    public void judge_ishabitable()
    {
        if(ishabitable)
            System.out.println("该星球适合人类居住");
        else
            System.out.println("该星球不适合人类居住");
    }
}


public abstract class Jovian_planet extends Planet{
    public String component = "气体";
}

public class Mercury extends Terrestrial_planet{
    Mercury(){
        this.set_name("Mercury");
        this.mass = 3.3011*1e23;
        this.radius = 2440;
        this.distance = 0.38;
        this.moon = false;
        this.ishabitable = false;
    }
    public void rotation()
    {
        System.out.println("水星自转");
    }
    public void print_relative_location()
    {
        System.out.println("水星是最靠近太阳的行星，离太阳"+distance+"天文单位");
    }
}

public class Venus extends Terrestrial_planet{
    Venus(){
        this.set_name("Venus");
        this.mass = 4.8675*1e24;
        this.radius = 6051.8;
        this.distance = 0.72;
        this.moon = false;
        this.ishabitable = false;
    }
    public void rotation()
```

```
    {
        System.out.println("金星自转");
    }
    public void print_relative_location()
    {
        System.out.println("金星在水星和地球之间，离太阳"+distance+"天文单位");
    }
}

public class Earth extends Terrestrial_planet{
    Earth(){
        this.set_name("Earth");
        this.mass = 5.97237*1e24;
        this.radius = 6371.0 ;
        this.distance = 1;
        this.moon = true;
        this.ishabitable = true;
    }
    public void rotation()
    {
        System.out.println("地球自转");
    }
    public void print_relative_location()
    {
        System.out.println("地球在金星和火星之间，离太阳"+distance+"天文单位");
    }
}

public class Mars extends Terrestrial_planet{
    Mars(){
        this.set_name("Mars");
        this.mass = 6.4185*1e23;
        this.radius =  3389.5 ;
        this.distance = 1.52;
        this.moon = true;
        this.ishabitable = false;
    }
    public void rotation()
    {
        System.out.println("火星自转");
    }
    public void print_relative_location()
    {
        System.out.println("火星在地球和木星之间，离太阳"+distance+"天文单位");
    }
}
```

```
public class Jupiter extends Jovian_planet{
    Jupiter(){
        this.set_name("Jupiter");
        this.mass = 1.8981*1e27;
        this.radius =  69911 ;
        this.distance = 5.20;
        this.moon = true;
    }
    public void rotation()
    {
        System.out.println("木星自转");
    }
    public void print_relative_location()
    {
        System.out.println("木星在火星和土星之间，离太阳"+distance+"天文单位");
    }
}

public class Saturn extends Jovian_planet{
    Saturn(){
        this.set_name("Saturn");
        this.mass = 5.6846*1e26;
        this.radius = 60268 ;
        this.distance = 9.54;
        this.moon = true;
    }
    public void rotation()
    {
        System.out.println("土星自转");
    }
    public void print_relative_location()
    {
        System.out.println("土星在木星和天王星之间，离太阳"+distance+"天文单位");
    }
}

public class Uranus extends Jovian_planet{
    Uranus(){
        this.set_name("Uranus");
        this.mass = 8.6810*1e25;
        this.radius = 25559;
        this.distance = 19.218;
        this.moon = true;
    }
    public void rotation()
```

```
    {
        System.out.println("天王星自转");
    }
    public void print_relative_location()
    {
        System.out.println("天王星在土星和海王星之间，离太阳"+distance+"天文单位");
    }

}

public class Neptune extends Jovian_planet{
    Neptune(){
        this.set_name("Neptune");
        this.mass = 1.0243*1e26;
        this.radius = 24764;
        this.distance = 30.06;
        this.moon = true;
    }
    public void rotation()
    {
        System.out.println("海王星自转");
    }
    public void print_relative_location()
    {
        System.out.println("海王星在 8 大行星中离太阳最远，离太阳"+distance+"天文单位");
    }

}

public class Pluto extends Dwarf_planet{
    Pluto()
    {
        this.set_name("Pluto");
        this.mass = 1.303*1e22;
        this.distance = new double[] {29.656, 49.319};
        this.radius = 1187;
        this.moon = true;
    }
    public void rotation()
    {
        System.out.println("冥王星在自转");
    }
    public void print_poor_Pluto()
    {
        System.out.println("国际天文联合会（IAU）在 2006 正式定义行星概念,将冥王星排除行星范围,
将其归类为矮行星（类冥天体）。");
    }
```

```
    public void print_relative_location()
    {
        System.out.println("由于冥王星的轨道高度倾斜，在小部分时候冥王星比海王星轨道更接近太阳，
其近日点据太阳"+distance[0]+"天文距离");
    }
}
```

2）图示这些类的继承关系？

利用在线绘图工具做 UML 类图以显示继承关系如下：

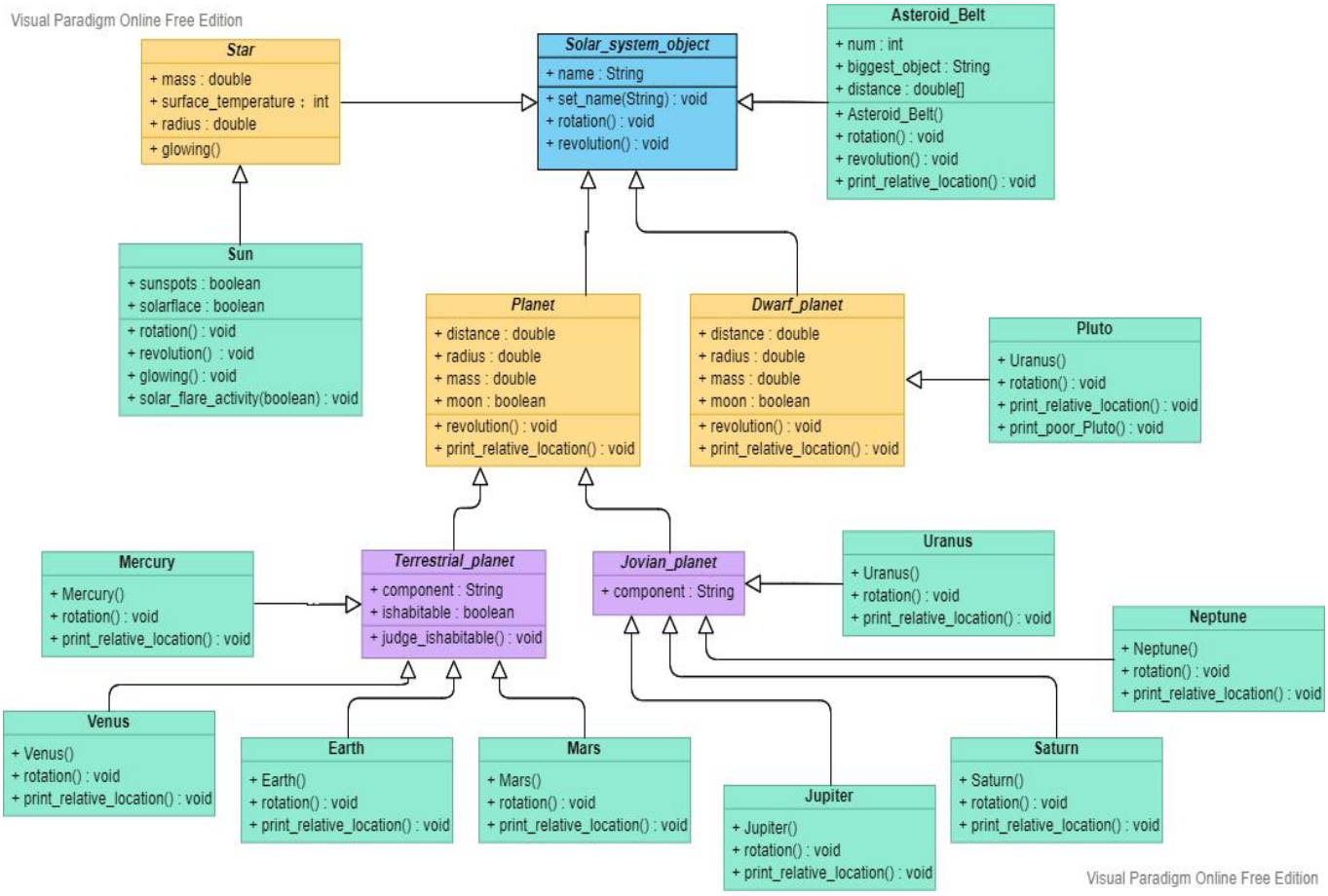


Fig 2 UML 类图-展示继承关系

（2）说明 Java 线程的活动周期以及在 Java 程序中如何创建线程？。

线程是指进程中单一顺序的执行流。线程共享同样的地址空间并共同构成一个大的进程。

每个线程都要经历创建、就绪、运行、阻塞和死亡 5 个状态，线程从产生到消失的变化状态如图所示：

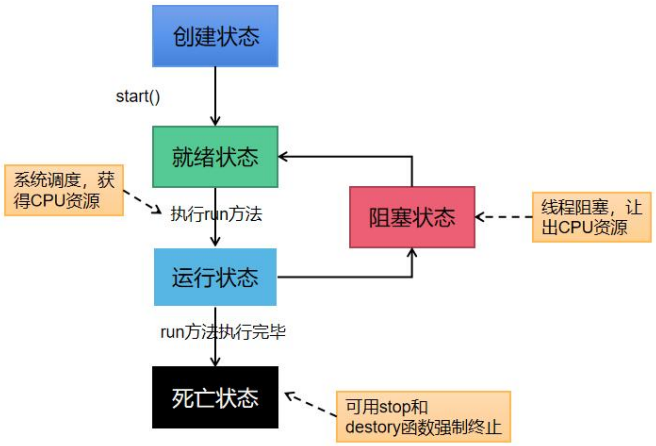


Fig 3 线程的活动周期

1. 创建状态

当通过 new 命令创建一个线程对象，则该线程对象就处于创建状态。例如，下面语句所示：

```
Thread thread1 = new Thread ();
```

创建状态是线程已被创建但未开始执行的一个特殊状态。此时线程对象拥有自己的内存空间，但没有分配 CPU 资源，需通过 start 方法进入就绪状态等待 CPU 资源。

2. 就绪状态

处于就绪状态的线程对象通过 start 方法进入就绪状态。例如，下面语句所示：

```
Thread thread1 = new Thread ();
thread1.start ();
```

start 方法同时调用了线程体，也就是 run 方法，表示线程对象正等待 CPU 资源，随时可被调用。

处于就绪状态的线程已被放到某一队列等待系统为其分配对 CPU 的控制权。至于何时可真正的执行，取决于线程的优先级以及队列当前状况。线程依据自身优先级进入等待队列的相应位置。如果线程的优先级相同，将遵循“先来先服务”的调度原则。如果某些线程具有较高的优先级，这些较高优先级线程一旦进入就绪状态，将抢占当前正在执行线程的 CPU 资源，这时当前线程只能重新在等待队列中寻找自己的位置，休眠一段时间，等待这些具有较高优先级的线程执行任务之后，被某一事件唤醒。一旦被唤醒，这些线程就开始抢占 CPU 资源。

优先级高的线程通常用来执行一些关键性和一些紧急任务，如系统事件的相应和屏幕显示，低优先级线程往往需要等待更长的时间才有机会运行。由于系统本身无法终止高优先级线程的执行，如果程序中用到了优先级较高的线程对象，那么可以不时让这些线程放弃对 CPU

资源的控制权。例如使用 sleep 方法，休眠一段时间，以使其他线程能有机会运行。

3. 运行状态

若线程处于正在运行的状态，表示线程已经拥有了对处理器的控制权，其代码目前正在运行，除非运行过程中控制权被另一优先级更高的线程抢占，否则这一线程将一直持续到运行完毕。

4. 阻塞状态

如果一个线程处于阻塞状态，那么该线程则无法进入就绪队列。处于阻塞状态的线程通常必须由某些事件唤醒。至于是何种事件，则取决于阻塞发生的原因。例如，处于休眠中的线程必须由阻塞固定的一段事件才能被唤醒；被挂起或处于消息等待状态的线程则必须由一外来事件唤醒。

5. 死亡状态

死亡状态（或终止状态），表示线程已退出运行状态，并且不再进入就绪队列。其原因可能是线程已执行完毕（正常结束）；也可能是该线程被另一线程强行中断，即线程自然撤销或被停止。自然撤销是从线程的 run 方法正常退出，即当 run 方法结束后，该线程自然撤销。调用 stop 方法可以强行停止当前线程（在 JDK2 中作废）。

总之，一个线程的生命周期一般经过如下步骤：

（1）一个线程通过 new()操作实例化后，进入创建状态。

（2）通过调用 start 方法进入就绪状态，一个处在就绪状态的线程将被调度执行，执行该线程相应的 run 方法中的代码。

（3）通过调用线程的（或从 Object 类继承过来的）sleep()或 wait()等方法，这个线程进入阻塞状态。一个线程也可能自己完成阻塞操作。

（4）当 run 方法执行完毕，或有其他终止条件产生，如执行 System.exit 等方法，则一个线程就进入死亡状态。

在 Java 语言中有以下方法创建线程方法以及简单示例：

（1）通过创建 Thread 类的子类来构造线程。Java 定义了一个直接从根类 Object 中派生的 Thread 类。所有从这个类派生的子类或间接子类，均为线程。

例：

```
class ThreadOne extends Thread //继承 Thread 类
{
  public void run() //重载 run 方法
  {
    System.out.println(currentThread().getName()+"Thread started:" );
```

```
    for(int i=0;i<6;i++){
       System.out.print(currentThread().getName()+":  i="+i+"\n");
    try{ sleep(500);}catch(InterruptedException e){}
    }
  }
}
```

（2）通过实现一个 Runnable 接口来构造线程。

例：

```
class ThreadCounting  implements Runnable  //实现接口
{
  public void run()
  {
    for(int i=0;i<10;i++){
       System.out.print(Thread.currentThread().getName()+":  i="+i+"\n");
        try{ Thread.sleep(200);}
        catch(InterruptedException e){System.out.print(e);}
    }
  }
}
```

（3）自学时发现，从 Java 5 开始，Java 提供了 Callable 接口，该接口是 Runnable 接口的增强版，Callable 接口提供了一个 call()方法，可以看作是线程的执行体，但 call()方法比 run()方法更强大。

- call()方法可以有返回值。
- call()方法可以声明抛出异常。

创建并启动线程的步骤如下：

1 创建 Callable 接口的实现类，并实现 call()方法，该 call()方法将作为该线程的执行体，且该 call()方法有返回值，再创建 Callable 的实例。从 Java 8 开始，可以直接使用 Lamda 表达式创建 Callable 对象。

2 使用 FutureTask 类来包装 Callable 对象，该 FutureTask 对象封装了该 Callable 对象的 call()方法的返回值。

3 使用 FutureTask 对象作为 Thread 对象的 target 创建并启动新线程。

4 调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值。

例：

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.FutureTask;
```

```
public class CallableThreadTest implements Callable<Integer> {
    public static void main(String[] args)
    {
        CallableThreadTest ctt = new CallableThreadTest();
        FutureTask<Integer> ft = new FutureTask<>(ctt);
        for(int i = 0;i < 100;i++)
        {
            System.out.println(Thread.currentThread().getName()+" 的循环变量 i 的值"+i);
            if(i==20)
            {
                new Thread(ft,"有返回值的线程").start();
            }
        }
        try
        {
            System.out.println("子线程的返回值：  "+ft.get());
        } catch (InterruptedException e)
        {
            e.printStackTrace();
        } catch (ExecutionException e)
        {
            e.printStackTrace();
        }

    }
    @Override
    public Integer call() throws Exception
    {
        int i = 0;
        for(;i<100;i++)
        {
            System.out.println(Thread.currentThread().getName()+" "+i);
        }
        return i;
    }
}
```

（3）举例说明 Java Socket 工作机制及 Java Socket 的主要 API 类有哪些？

Socket 被翻译为套接字，是网络通信的一种底层编程接口。在 TCP/IP 通信协议中，套接字（Socket）就是 IP 地址与端口号的组合。

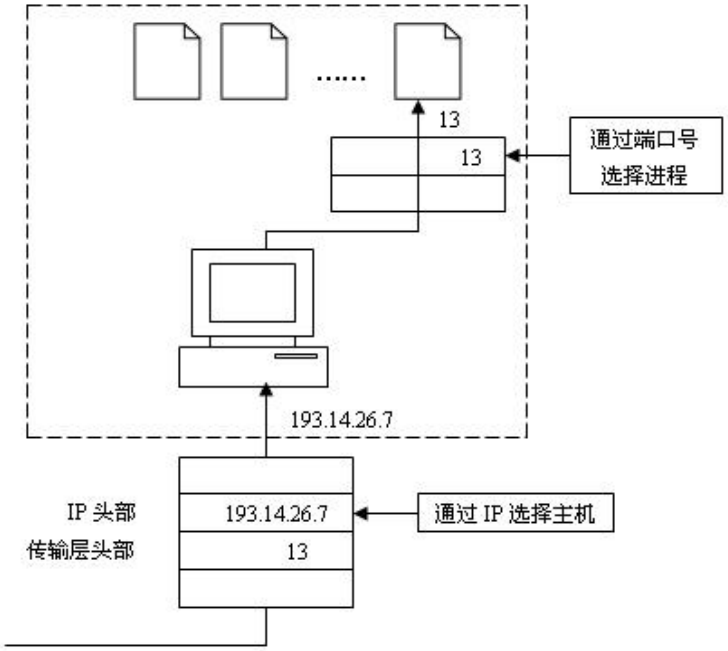


Fig 4 IP 地址 193.14.26.7 与端口号 13 组成一个套接字(图片来自《Java 语言程序设计》第三版，张思民著)

当两个网络程序进行通信时，它们可以通过使用 Socket 类建立套接字连接。一个典型的网络程序通讯就是一对客户机和服务器间的通讯。呼叫方称为“客户端”，负责监听的一方称为“服务器端”。

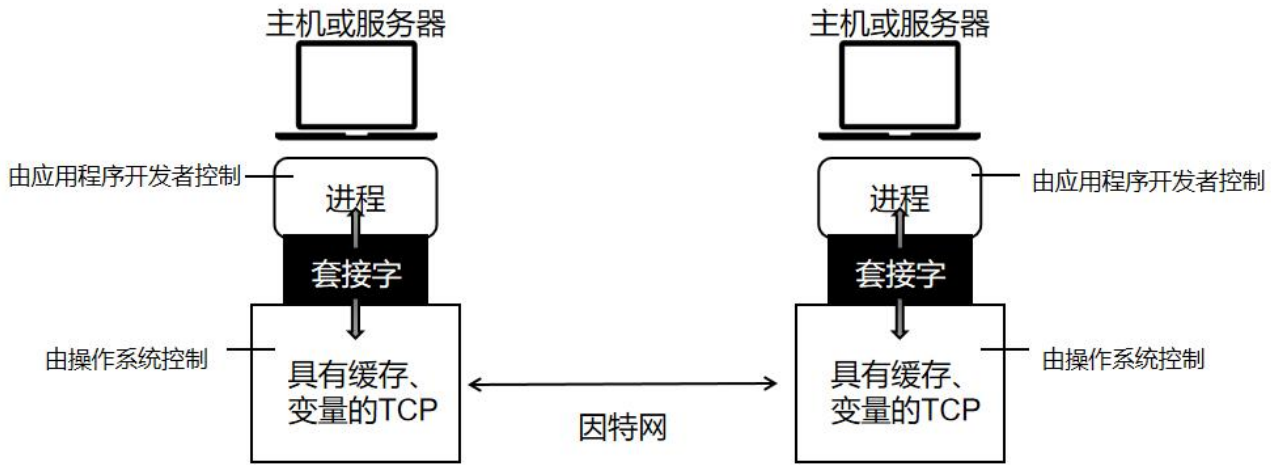


Fig 5 应用进程、套接字和下面的以 TCP 为例的传输层协议(ref:《计算机网络 自顶而下方法》第二章 p58)

比如说，将进程类比做一座房子，而它的套接字可以类比于它的门。当一个进程想向位于另外一台主机上的另一个进程发送报文(message)时，它将报文推出该门（套接字）。该发送进程规定该门到另外一侧之间有运输的基础设施，该设施将把报文传送到目的进程的门口。一旦该报文抵达目的主机，它通过接收进程的门（套接字）传递，然后接收进程对该报文进行处理。

应用 Socket 机制的网络通信又分为基于 TCP 的流式通信和基于 UDP 的数据报通信。TCP

协议通过 socket 套接字建立一条虚电路，而 UDP 数据报的每个数据包要包含目的地址和端口号。

Java Socket 的主要 API 类有：

类名	说明
java.net.Socket	客户端套接字
java.net.ServerSocket	服务器套接字
java.net.InetAddress	用来表示 IP 地址的高级表示
java.net.InetSocketAddress	实现 IP 套接字地址（IP 地址 + 端口号）
java.net.DatagramSocket	用于发送和接收数据报包的套接字。
java.net.DatagramPacket	表示数据报包

以 TCP 为例：

套接字使用 TCP 提供了两台计算机之间的通信机制。 客户端程序创建一个套接字，并尝试连接服务器的套接字。当连接建立时，服务器会创建一个 Socket 对象。客户端和服务器现在可以通过对 Socket 对象的写入和读取来进行通信。

java.net.Socket 类代表一个套接字，并且 java.net.ServerSocket 类为服务器程序提供了一种来监听客户端，并与他们建立连接的机制。

以下步骤在两台计算机之间使用套接字建立 TCP 连接时会出现：

- 服务器实例化一个 ServerSocket 对象，表示通过服务器上的端口通信。
- 服务器调用 ServerSocket 类的 accept() 方法，该方法将一直等待，直到客户端连接到服务器上给定的端口。
- 服务器正在等待时，一个客户端实例化一个 Socket 对象，指定服务器名称和端口号来请求连接。
- Socket 类的构造函数试图将客户端连接到指定的服务器和端口号。如果通信被建立，则在客户端创建一个 Socket 对象能够与服务器进行通信。
- 在服务器端，accept() 方法返回服务器上一个新的 socket 引用，该 socket 连接到客户端的 socket。

连接建立后，通过使用 I/O 流在进行通信，每一个 socket 都有一个输出流和一个输入流，客户端的输出流连接到服务器端的输入流，而客户端的输入流连接到服务器端的输出流。

TCP 是一个双向的通信协议，因此数据可以通过两个数据流在同一时间发送。

（4）如何理解 Java 程序中的“接口”？举例说明如何声明和使用“接口”？

接口是抽象类的一种，只包含常量和方法的定义，而没有变量和具体方法的实现，且其方法都是抽象方法。它的用处体现在下面几个方面：

1）通过接口实现不相关类的相同行为，而无须考虑这些类之间的关系。

2）通过接口指明多个类需要实现的方法

3）通过接口了解对象的交互界面，而无需了解对象所对应的类

接口的定义一般包括接口声名和接口体，定义为：

```
[public] interface 接口名[extends 父接口名]
{
    ...//接口体
}
```

extends 子句与类声明的 extends 子句基本相同；不同的是一个接口可有多个父接口，用逗号隔开，而一个类只能有一个父类。在类的声明中用 implements 子句来表示一个类使用某个接口，在类体中可以使用接口中定义的常量，而且必须实现接口中定义的所有方法。

举例，鸟和飞机都可以飞行，但具体飞行的方式不同，在接口中定义飞行方法，并分别对两类调用飞行方法进行打印：

```
interface Fly //定义接口
{
    public void fly();
}
class Plane implements Fly //定义实现接口的 plane 类
{
    String name;
    double speed;
    public void set_inf(String name, double speed)
    {
        this.name = name;
        this.speed = speed;
    }
    public void fly() //编写实现接口 fly()方法的具体代码
    {
        System.out.println(name+"正在以"+speed+"马赫飞行");
    }
}
class Bird implements Fly //定义实现接口的 bird 类
{
    public void fly() //编写实现接口 fly()方法的具体代码
```

```
    {
        System.out.println("鸟在飞");
    }
}
public class InterfaceTest {
    public static void main(String args[])
    {
        Plane plane = new Plane();
        Bird bird = new Bird();
        plane.set_inf("波音 737", 0.8);
        plane.fly();
        bird.fly();
    }
}
```

打印结果：

```
波音 737 正在以 0.8 马赫飞行

鸟在飞
```

二．实践题

1. 项目描述：题目名称及背景概况。

题目名称：Microsoft Access 数据据库 .mdb 文件查看工具

背景概况：数据库是管理和组织信息和数据的综合系统。作为关系型数据库，Access 以表为单位组织数据并支持使用 SQL（结构化查询语言）进行增、删、改、查等操作。由于 Java 的跨平台可移植性，使用 Java 开发的数据库系统可以充分利用 Internet 上的各种软、硬件资源，从而大大扩展了适用性，并降低了升级和维护的费用，常常被作为开发的有利工具。本数据库查看系统作为学生作业，主要体现了应用 JDBC 的数据库操作和 Swing 图形界面设计的实际应用，同时根据所学，在基本页面基础上进行了功能和页面的丰富和扩展。可以实现.mdb 数据库文件查看的同时，功能较为完成，页面丰富美观，在一定程度上简化了数据库编辑和统计难度，同时附带了说明文档，使用户可以清晰理解页面功能。另一方面，本系统开发参考了面向对象设计模式的思想，对软件工程体现了一定理解，在完成基本功能的同时体现了清晰的设计思想。

2. 项目分析：项目的目的、用途或需要解决的问题以及需求。

项目的主要目的在于开发一独立程序用于查看 Microsoft Access 文档。该项目有以下需求：

1）程序能够连接 Access 数据库

2）程序主页面可以可视化 Access 数据库的表名列表，双击表名列表可以打开 Jtable 可视化的

数据表

3）程序打开数据表后，可以通过菜单对话框和 JDBC 连接 Access 数据库对数据表中的数据进行增加、删除、修改的操作，完成操作后更新表格显示。

4）程序打开数据表后，可以通过菜单对话框和 JDBC 连接数据库对数据库中的数据进行查询操作，进行查询操作后用文本框显示查询结果。

5）使用的库的要求为：有关数据库操作用 JDBC 编写；用户界面用 Swing 实现。

6）考虑到后续可能的扩展开发，添加了登录页面以便于后续可以添加用户管理，多数据库管理等功能。

3. 设计：功能设计、界面设计、数据结构和数据库设计、对象和类设计、文件系统目录结构设计。

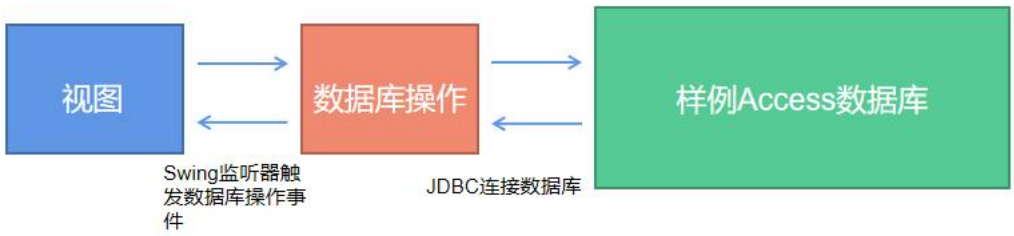


Fig 6 基本设计展示

功能设计：

- 用户打开.jar 文件，显示登录页面
- 用户点击使用说明按钮，打开 readme.md 文档可查看程序使用说明
- 用户点击登录按钮，打开主页面显示数据库表名列表
- 双击主页面表名，可打开数据表，显示表格页面
- 单击主页面菜单对应操作，可弹出对话框，输入对应操作语句可以进行操作，若输入非法，则会弹窗提示发生错误，其中可进行操作如下：

- 单击 Add，弹出对话框，可输入指令后按下确认，对数据表进行添加操作

- 单击 Delete，弹出对话框，可输入指令后按下确认，对数据表进行删除操作

- 单击 Update，弹出对话框，可输入指令后按下确认，对数据表进行修改操作

- 单击 Query，弹出对话框，可输入指令后按下确认，查询某一数据（默认查询范围为全表），对数据表进行查询操作并将返回结果打印在文本框中。

- 点击主页面菜单中关于作者项，可弹出页面显示作者学号姓名联系方式等相关信息

页面设计：

1. 登录界面

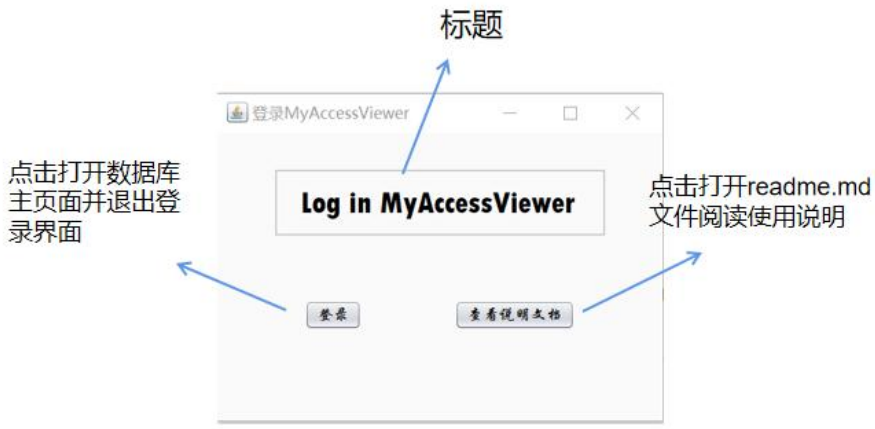


Fig 7 登录页面：利用 Frame 类并设置了颜色和字体

2. 主页面

- 表显示页面

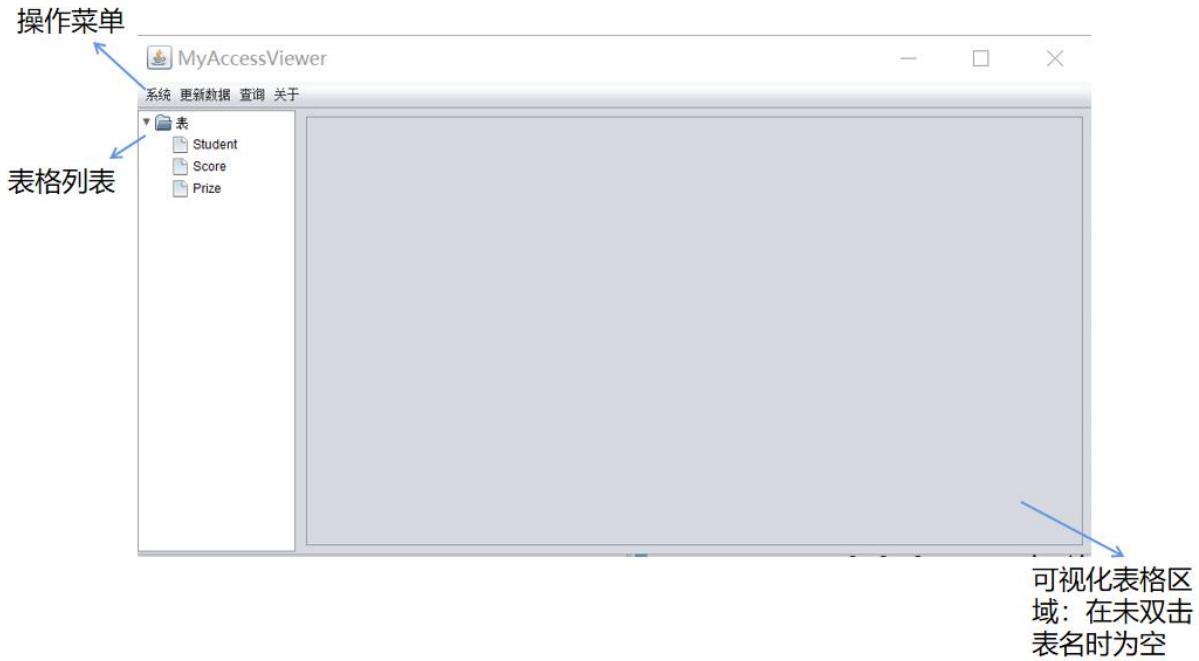


Fig 8 表显示页面：初始页面

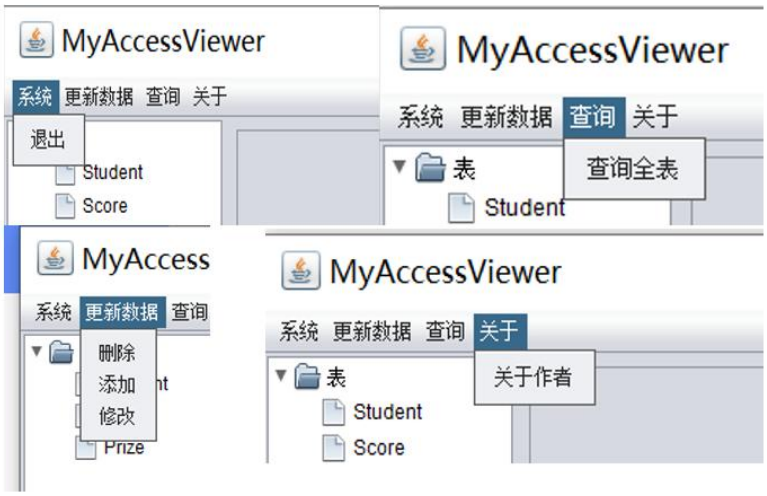
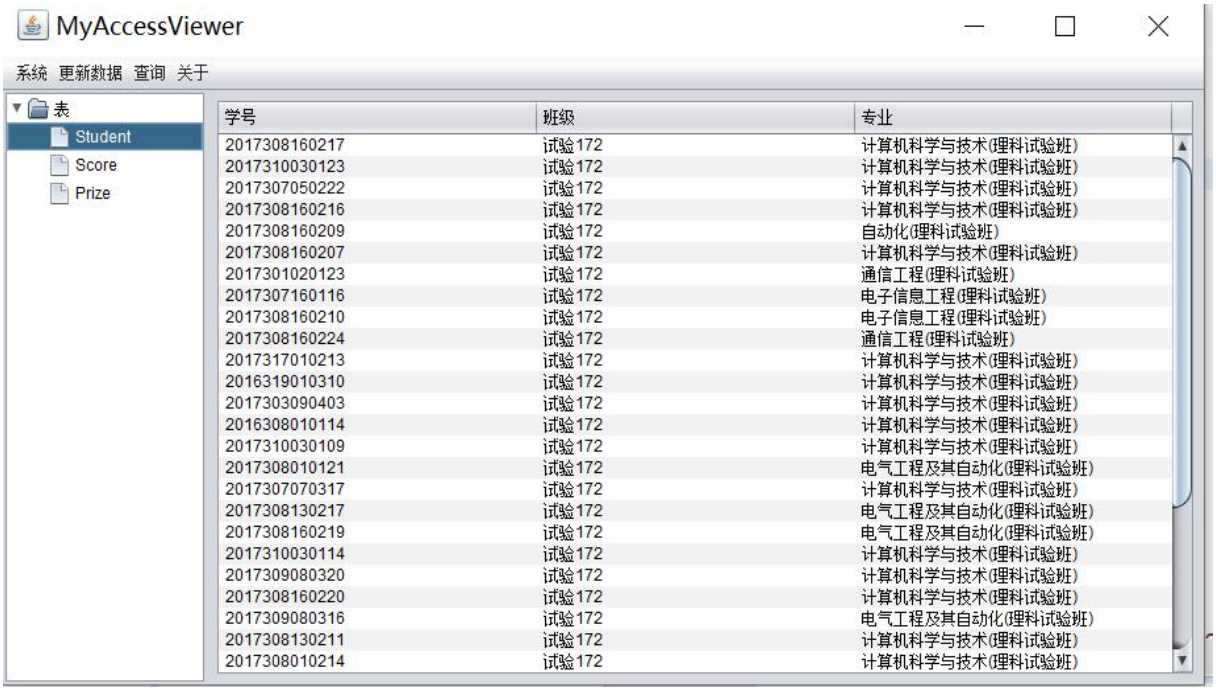


Fig 9 表显示页面：菜单详情

双击表名后，可触发事件显示表格



学号	班级	专业
2017308160217	试验172	计算机科学与技术(理科试验班)
2017310030123	试验172	计算机科学与技术(理科试验班)
2017307050222	试验172	计算机科学与技术(理科试验班)
2017308160216	试验172	计算机科学与技术(理科试验班)
2017308160209	试验172	自动化(理科试验班)
2017308160207	试验172	计算机科学与技术(理科试验班)
2017301020123	试验172	通信工程(理科试验班)
2017307160116	试验172	电子信息工程(理科试验班)
2017308160210	试验172	电子信息工程(理科试验班)
2017308160224	试验172	通信工程(理科试验班)
2017317010213	试验172	计算机科学与技术(理科试验班)
2016319010310	试验172	计算机科学与技术(理科试验班)
2017303090403	试验172	计算机科学与技术(理科试验班)
2016308010114	试验172	计算机科学与技术(理科试验班)
2017310030109	试验172	计算机科学与技术(理科试验班)
2017308010121	试验172	电气工程及其自动化(理科试验班)
2017307070317	试验172	计算机科学与技术(理科试验班)
2017308130217	试验172	电气工程及其自动化(理科试验班)
2017308160219	试验172	电气工程及其自动化(理科试验班)
2017310030114	试验172	计算机科学与技术(理科试验班)
2017309080320	试验172	计算机科学与技术(理科试验班)
2017308160220	试验172	计算机科学与技术(理科试验班)
2017309080316	试验172	电气工程及其自动化(理科试验班)
2017308130211	试验172	计算机科学与技术(理科试验班)
2017308010214	试验172	计算机科学与技术(理科试验班)

Fig 10 表显示界面：显示表格，以 Student 表为例

同理双击其他表名也可以显示其他表的内容。

- 数据库操作交互窗口——更新操作

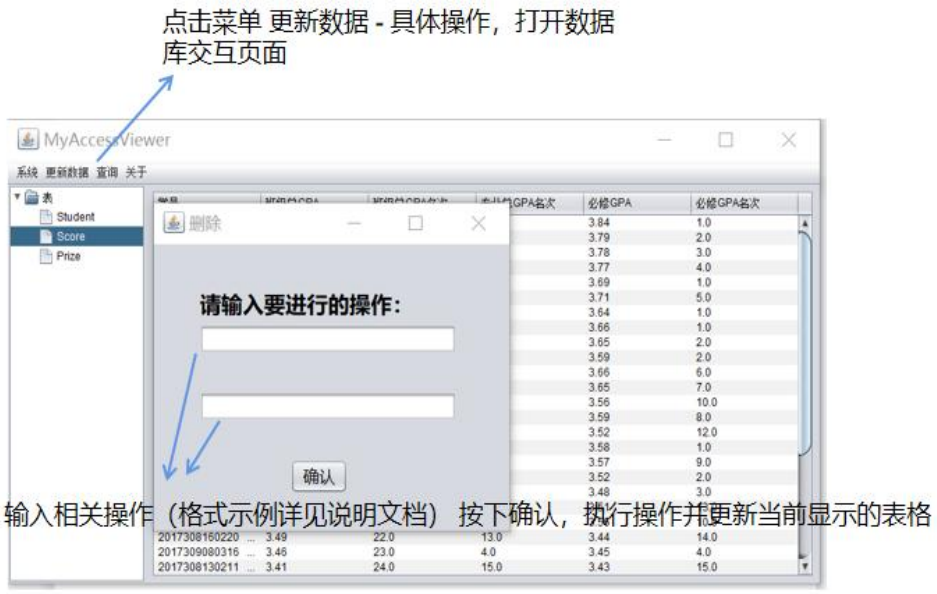


Fig 11 数据库操作交互窗口：更新数据

Fig 12 数据库操作交互窗口：查询数据

3. 作者说明页面

为便于软件使用者与开发者交互，本软件设计了作者说明界面。点击主页面菜单-关于-关于作者，打开关于作者页面如下：

Fig 13 作者说明页面

数据结构与数据库设计：

该软件的主要内存占用和时间复杂度主要涉及增删改查和查看表格过程中数据表的存储，本软件开发利用 java.util.Vector 包，获取查询获得的 ResultSet 对象后创建二维动态数组，利用一个一重循环存储每行数据。在这里使用二维动态数组的优点为：1. 可以实现不指定一块内存大小的数据的连续存储，在读取不同表时可以实现自适应而不必受预先设置的空间大小所限 2.在一定程度上节省空间，对于小对象效率更高。3.便于开发，可以直接用于初始化 jtable 画出表格

为展示此.mdb 文件查看工具功能，本程序附带一示例数据库 AccessDatabase.mdb，主要用于展示中国农业大学信电学院试验 172 班的成绩和奖学金获得情况(数据来源中国农业大学信息与电气工程学院官网公示)，其中共包括三个表：

数据库表名	关系模式名称	备注
Student	学生	学生信息表
Score	分数	学生成绩表
Prize	奖励	学生获奖表

学生基本情况数据表，其数据类型如下：

字段名	字段类型	约束控制
学号	文本	主键
班级	文本	非空
专业	文本	非空

学生成绩表，其数据类型如下：

字段名	字段类型	约束控制
学号	文本	主键
班级总 GPA	数字-常规数字	非空
班级总 GPA 名次	数字-常规数字	非空
专业总 GPA 名次	数字-常规数字	非空
必修 GPA	数字-常规数字	非空
必修 GPA 名次	数字-常规数字	非空

学生获奖表，其数据类型如下：

字段名	字段类型	约束控制
学号	文本	主键
姓名	文本	非空
综合/专项	文本	非空
单项	文本	非空

对象和类设计

主要类设计：

数据库操作层面：

- MyBusiness 类 主要包含了基本的数据库操作方法和数据库 Connection 和 Statement 属性

可视化页面层面：

- MyMainFrame 类 继承 Frame 类，包含可滑动面板，菜单、树和表等 Swing 控件和事件监听方法以及可视化数据库表内容方法和 MyBusiness 类对象 用于可视化主页面
- MyLogin 类 继承 Frame 类，包含面板，标签，按钮等 Swing 对象和事件监听方法 用于可视化登录页面
- MyUpdateDialog 继承 Frame 类，包含文本框，按钮等 Swing 对象和事件监听方法和 MyMainFrame 属性 用于可视化数据库交互页面
- AboutMe 类 继承 Frame 类，包含很多标签对象 用于可视化关于作者页面

具体方法和属性实现可详见 4.主要功能或算法描述及流程中类的 UML 图所示

文件系统目录结构设计

MyAccessViewer 主项目

MyAccessViewer.jar 是整个项目打成的 jar 包,直接双击运行，打开登录界面。

MyAccessViewer2 项目文件夹

.setting 目录 eclipse 生成的元数据目录

bin 工程输出路径，存放了编译生成的.class 文件

src 源代码文件夹

MyAccessConnection 文件夹

MyBusiness.java 为具体事务操作（连接数据库，获取、增加、删改、查询记录等）

MyVisualization 文件夹

AboutMe.java 为作者说明界面

MyLogin.java 为登录界面

MyMainFrame.java 为主页面，可显示表格和相关数据

UpdateDialog.java 为操作对话框，输入语句后进行相关操作更新数据库表格显示或弹出查询结果

.classpath 和.project 为 eclipse 生成的工程描述文件

Access 文件夹

AccessDatabase.mdb 数据库，使用时请先将其设为数据源

readme.md 使用说明文档，在登录界面点击使用说明打开

4.主要功能或算法描述及流程。

本软件所实现的主要功能，是 Access 数据库的查看与增、删、改、查操作，并添加了软件登录和联系作者页面以完善软件作用。

- 数据库查看功能，通过 Mybusiness 类实现数据库操作，MyMainFrame 类实现可视化，其流程如下：

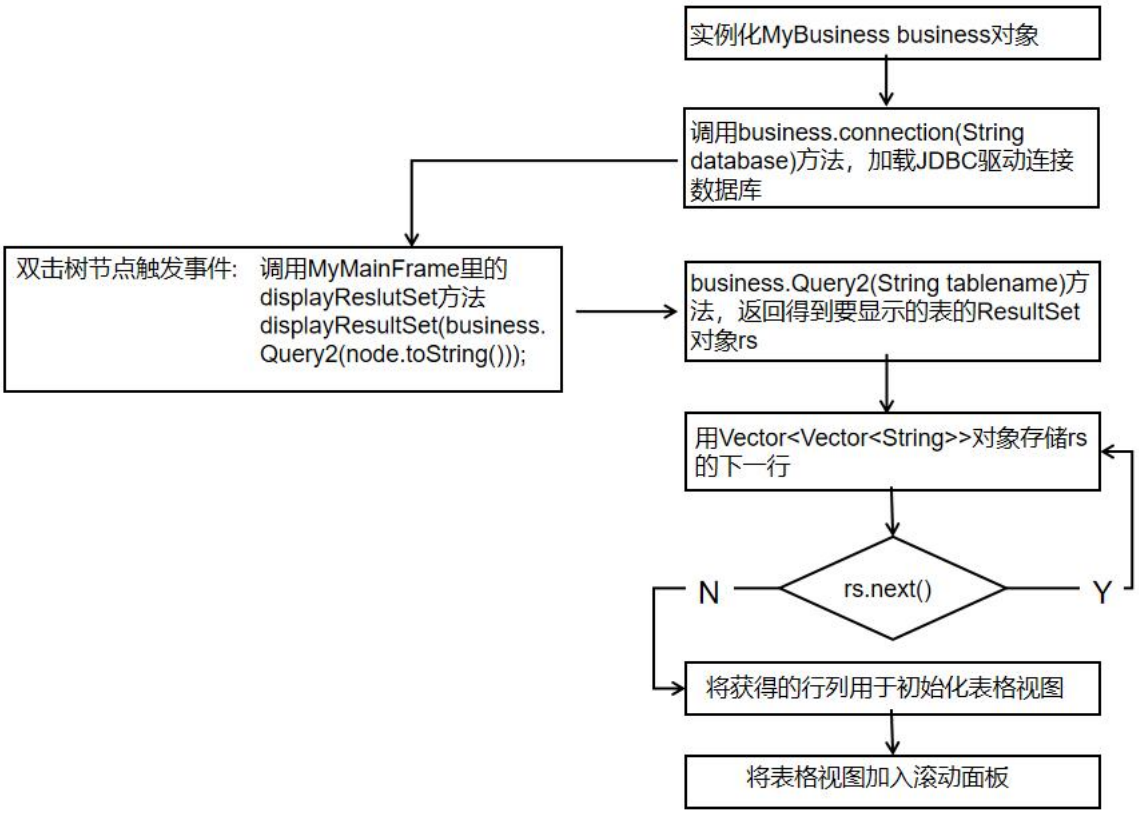


Fig 14 可视化表格流程图

- 数据库增加记录功能，通过 Mybusiness 类实现数据库操作，UpdateDialog 类实现与用户交互
- 数据库删除记录功能，通过 Mybusiness 类实现数据库操作，UpdateDialog 类实现与用户交互
- 数据库修改记录功能，通过 Mybusiness 类实现数据库操作，UpdateDialog 类实现与用户交互
- 数据库查询记录功能，通过 Mybusiness 类实现数据库操作，UpdateDialog 类实现与用户

交互

其流程如下所示：

Fig 15 数据库操作流程

为展示具体实现的类和包关系、属性和方法，利用 eclipse 插件 ModelGoon 和 ppt 绘制包依赖关系图，类图，类交互图如下：

1. 包依赖关系

本软件采用视图和数据库操作分离的设计，两个包 MyAccessConnection 和 MyVisualization 关联和功能如下：

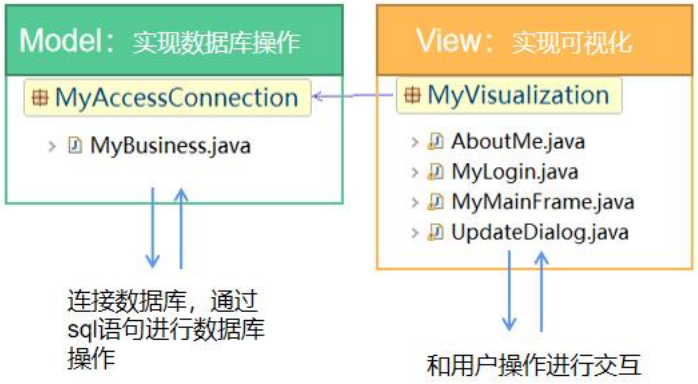


Fig 16 包依赖关系

2. 类交互关系

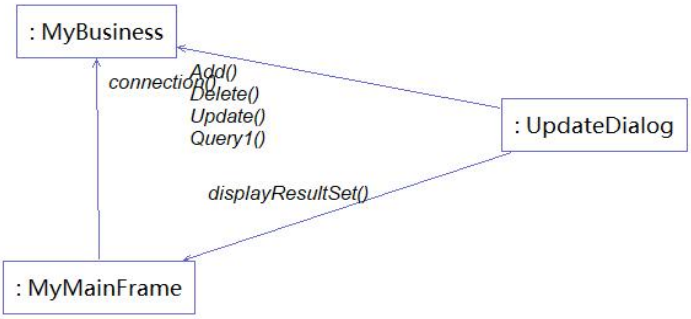


Fig 17 类交互关系

3. UML 类图

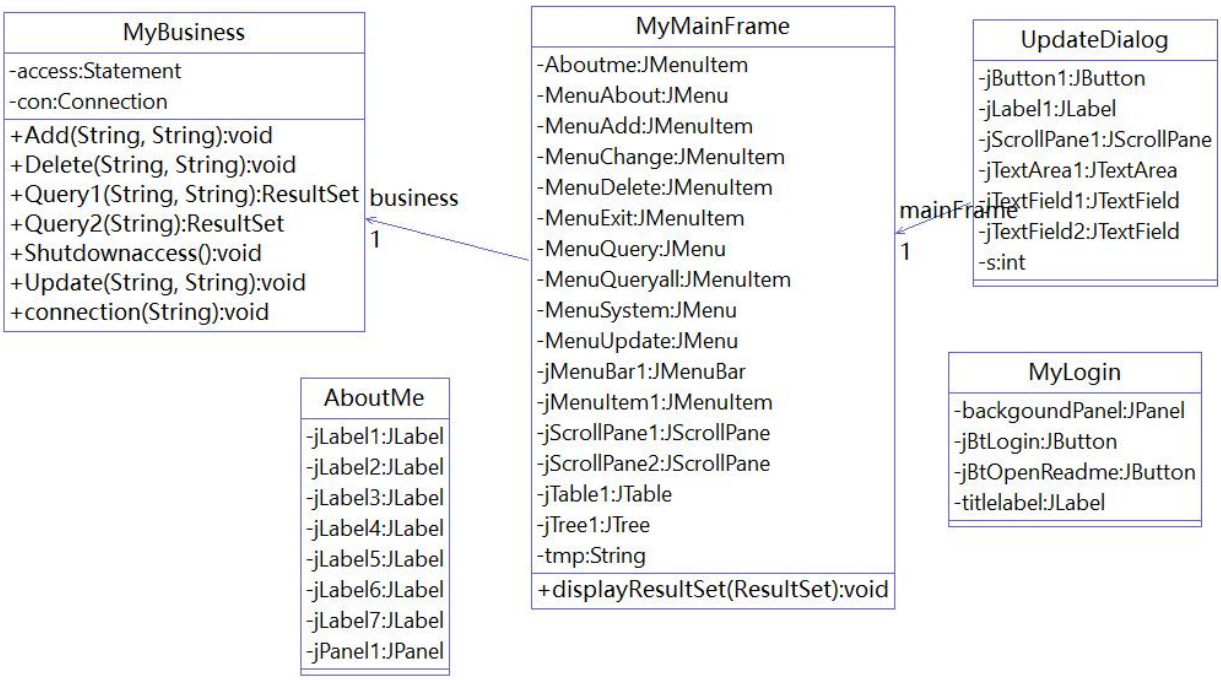


Fig 18 UML 类图

5. 文献资料

《关于 JDK8 连接 Access 的解决办法》， https://blog.csdn.net/cqt00/article/details/51580540 2021.7.21

《数据库系统概念》，中文第六版，Abraham Silberschatz, Henry F.Korth, S.Sudarshan 著，杨东青，李红燕，唐世渭等译

《Java 语言程序设计》，第三版及课程 ppt，张思民编著，清华大学出版社

《Java 语言程序设计》，第四版，李尊朝，苏军，李昕怡编著，中国铁道有限公司

Trail: Creating a GUI With Swing https://docs.oracle.com/javase/tutorial/uiswing/ 2021.8.3

《JAVA 图形界面（GUI）之表格》，https://blog.csdn.net/qq_34908844/article/details/79890062 2021.8.3

WHAT IS UML，https://www.uml.org/what-is-uml.htm

《软件工程导论》，张海藩，清华大学出版社

【Java 学习笔记】53：JTree 的使用(新选中事件和结点双击事件)， https://blog.csdn.net/SHU151218566/article/details/79269450 2021.8.3

《JAVA 将 ResultSet 结果集遍历到 List 中》，https://blog.csdn.net/HeatDeath/article/details/79559614 2021.8.7

《jdbc 实现学生管理系统》，https://juejin.cn/post/6876706025173745678 2021.8.8

6. 简要使用说明

6.1 支持环境和软件

本软件在 64 位 Windows 系统下开发，使用 Eclispe 编译器，Java 环境如下：

java version "1.7.0_80"

Java(TM) SE Runtime Environment (build 1.7.0_80-b15)

Java HotSpot(TM) 64-Bit Server VM (build 24.80-b11, mixed mode)

由于 Java 从 JDK1.8 之后就移除了 Access 桥接驱动，若使用 JDK1.8 及以上版本的 Java 运行时，将发生错误无法进行数据库操作，弹窗提示数据库驱动错误。

使用的本机数据库版本为 Microsoft Access 2010，所有功能均经过本机测试能正常使用。

如遇中文乱码问题，请检查编码格式是否统一。

6.2 使用方法

MyAccessViewer 主项目

MyAccessViewer.jar 是整个项目打成的 jar 包,直接双击运行，打开登录界面。

MyAccessViewer2 项目文件夹

.setting 目录 eclipse 生成的元数据目录

bin 工程输出路径，存放了编译生成的.class 文件

src 源代码文件夹

MyAccessConnection 文件夹

MyBusiness.java 为具体事务操作（连接数据库，获取、增加、删改、查询记录等）

MyVisualization 文件夹

AboutMe.java 为作者说明界面

MyLogin.java 为登录界面

MyMainFrame.java 为主页面，可显示表格和相关数据

UpdateDialog.java 为操作对话框，输入语句后进行相关操作更新数据库表格显示或弹出查询结果

.classpath 和.project 为 eclipse 生成的工程描述文件

Access 文件夹

AccessDatabase.mdb 数据库，使用时请先将其设为数据源

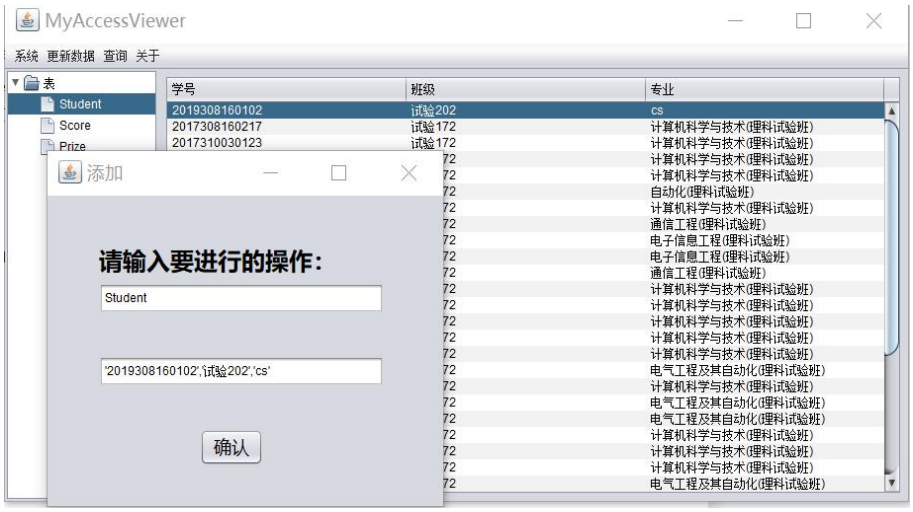
readme.md 使用说明文档，在登录界面点击使用说明打开

在数据源已设置的情况下，双击.jar 文件，打开登录界面，点击登录，打开主页面，双击表格名打开表格，单击菜单栏相关操作可打开交互对话窗进行增删改查操作。

增加记录格式为：

表名+值

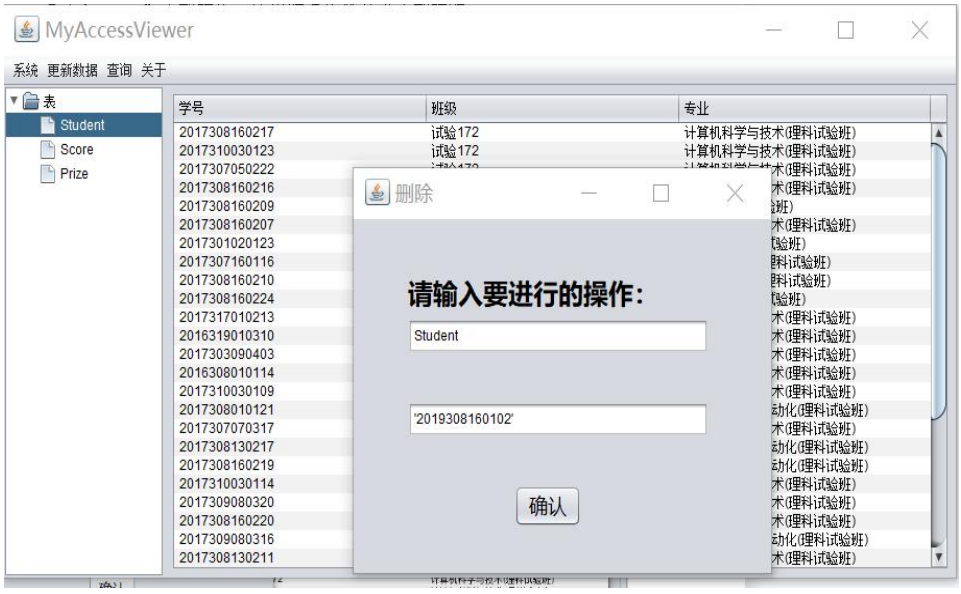
如：Student+'2019308160102','试验 202','cs'



删除记录格式为：

表名+主键

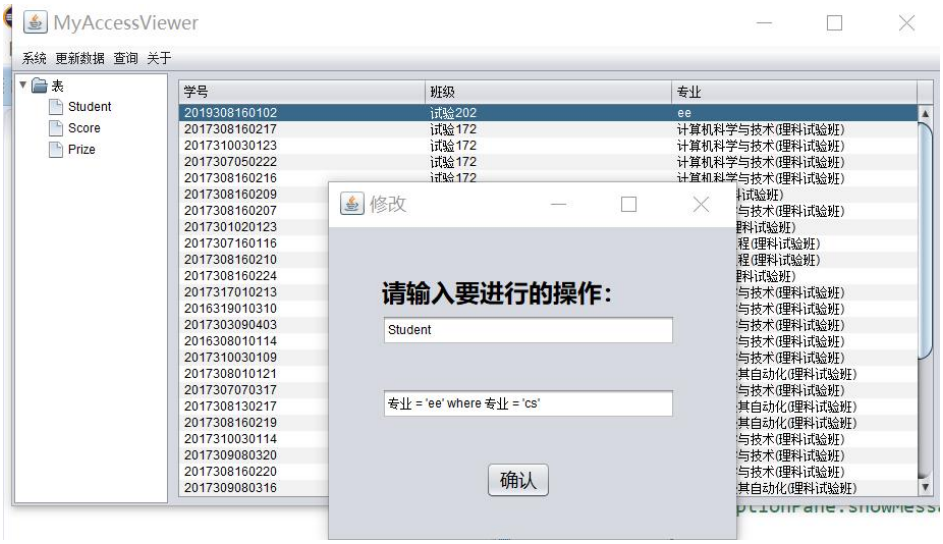
如：Student+'2019308160102'



修改记录格式为：

表名+修改值和条件

如：Student+专业 = ’ee’ where 专业 = ’cs’



查询记录格式为：

表名+查询条件

如：Student+学号=’2019308160102’