来谈谈JAVA面向对象 - 鲁班即将五杀，大乔送他回家？？

开发IDE为Eclipse或者MyEclipse。

首先，如果我们使用面向过程的思维来解决这个问题，就是第一步做什么，第二步做什么？

鲁班即将五杀，大乔送他回家

这个现象可以简单地拆分为两步，代码大概是这个样子的:

```
public class Test01 {

    public static void main(String[] args) {
        System.out.println("鲁班即将五杀");
        System.out.println("大乔送他回家");
    }

}
```

面向过程的思维大概就是这样。

如果我们用面向对象的思维来看待这个问题，首先，得抽象出有哪几个对象。

鲁班类：

```
/**
 * 鲁班类
 * @author Administrator
 *
 */
public class Luban {

    private String name = "鲁班"; //英雄的名字
    private int killCount = 0;    //击杀个数



}
```

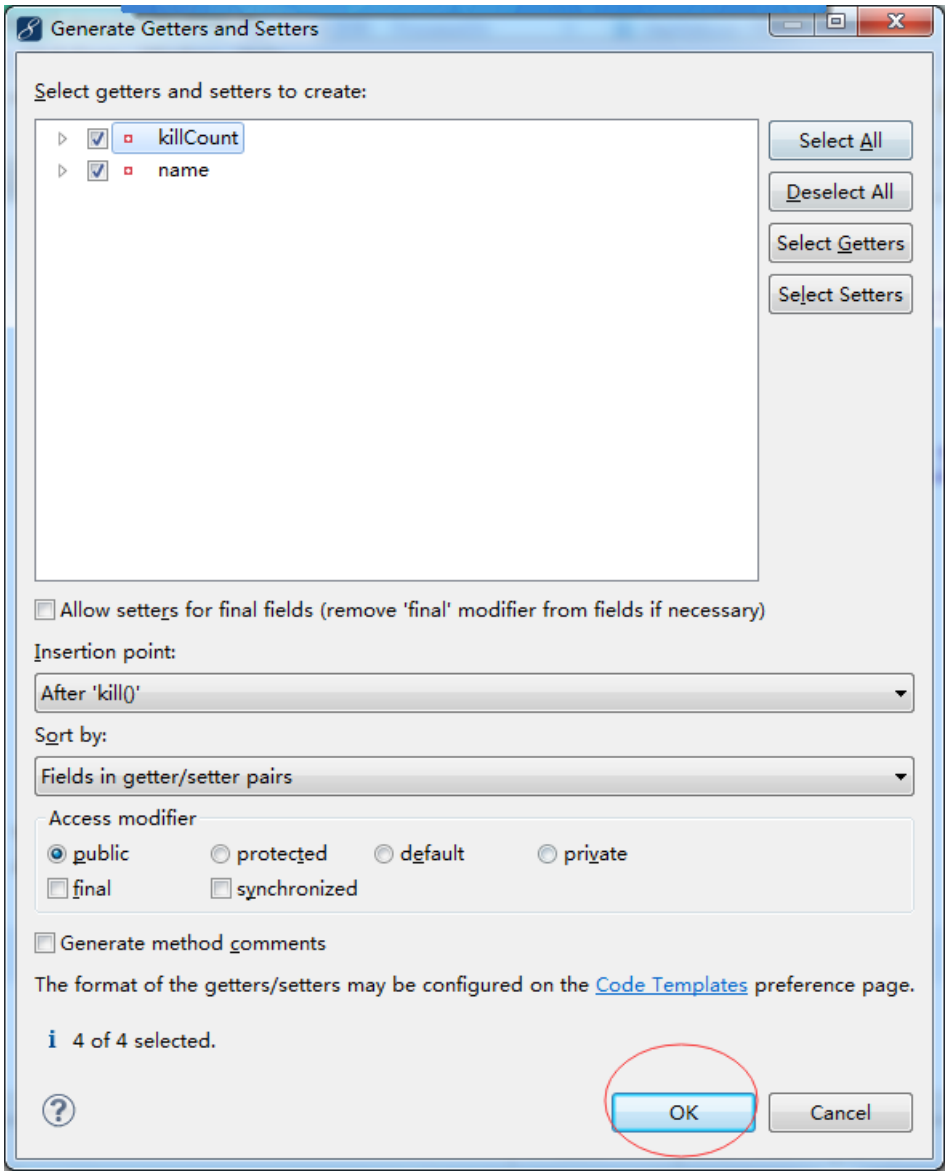
因为鲁班会有一个从 **first blood** 到 **penta kill** 的过程 , 在这个过程中，需要对killCount 这个变量一步步递增，所以，我们给它再加一个kill方法。

```
public void kill(){
    killCount++;

    switch (killCount) {
    case 1:
        System.out.println("First Blood!");
        break;
    case 2:
        System.out.println("Double Kill!");
        break;
    case 3:
        System.out.println("Triple kill!");
        break;
    case 4:
        System.out.println("Quadra kill!");
```

```
            break;
        case 5:
            System.out.println("Penta kill!");
            break;

        default:
            break;
        }

}
```

这个时候，我们发现，访问不了私有变量。我们需要给name和killCount添加对应的get，set方法：

Alt + S --> Generator Getters and Setters -->



这样就自动生成了！

至此，鲁班类告一段落，接下来编写大乔类：

```
/**
 * 大乔类
 * @author Administrator
 *
 */
public class Daqiao {

    private String name = "大乔";

    /**
     * 放大招的方法
     */
    public void DaZhao(Luban luban){

    }
```

```
}
```

大招方法需要把鲁班给装进去，大家思考如何实现送鲁班回家的过程？

这边我提供一个思路，给鲁班添加一个私有的布尔属性 isAtHome，默认是false。当鲁班被传递进大乔的大招方法里之后，就改为true。

```
private boolean isAtHome = false; //是否回泉水?
```

别忘了生成 get set 方法。

于是乎，大招方法就变成了这样：

```
/**
 * 放大招的方法
 */
public void DaZhao(Luban luban){
    luban.setAtHome(true);
}
```

现在大家再想一个问题，当鲁班回泉水了，还能不能继续调用 kill 方法？

肯定是不能了，修改后的 kill 方法，在killCount++之前就应该return掉：

```
public void kill(){

    if(this.isAtHome){
        setName("鲁班七号");
        System.out.println(name + ":我是谁，在干什么？？");
        return;
    }

    killCount++;

    switch (killCount) {
    case 1:
        System.out.println("First Blood!");
        break;
    case 2:
        System.out.println("Double Kill!");
        break;
    case 3:
        System.out.println("Triple kill!");
        break;
    case 4:
        System.out.println("Quadra kill!");
        break;
    case 5:
        System.out.println("Penta kill!");
        break;

    default:
        break;
    }

}
```

测试：

```
import bean.Daqiao;
import bean.Luban;

public class Test02 {

    public static void main(String[] args) {
        Luban luban = new Luban();
        luban.kill();
        luban.kill();
        luban.kill();
        luban.kill();

        Daqiao dq = new Daqiao();

        dq.DaZhao(luban);

        luban.kill();
```

```
    }
}
```

以上就是我们面向对象的一般思路，先抽象出有几个类，然后设计每个类中有哪些方法？

面向对象有三大特性，分别为继承，封装和多态。

继承

还是这个例子，思考：鲁班和大乔都属于王者峡谷里面的英雄。他们有很多共同的特性。我们可以抽象出一个通用的英雄类，Hero类 。

```
package bean;

/**
 * 英雄类
 * @author Administrator
 *
 */
public class Hero {

    protected String name;
    protected int killCount = 0;
    protected boolean isAtHome = false;

    public void kill(){

    }

    public void DaZhao(){

    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getKillCount() {
        return killCount;
    }

    public void setKillCount(int killCount) {
        this.killCount = killCount;
    }

    public boolean isAtHome() {
        return isAtHome;
    }

    public void setAtHome(boolean isAtHome) {
        this.isAtHome = isAtHome;
    }


}
```

然后让鲁班和大乔都继承这个类。

对鲁班而言，继承了Hero类以后，只需要重写kill即可，省去了很多代码：

```
package bean;

/**
 * 鲁班类
 * @author Administrator
 *
 */
public class Luban extends Hero{

    public void kill(){

        if(this.isAtHome){
            System.out.println(name + ":我是谁，在干什么？？");
            return;
        }
```

```
        killCount++;

        switch (killCount) {
        case 1:
            System.out.println("First Blood!");
            break;
        case 2:
            System.out.println("Double Kill!");
            break;
        case 3:
            System.out.println("Triple kill!");
            break;
        case 4:
            System.out.println("Quadra kill!");
            break;
        case 5:
            System.out.println("Penta kill!");
            break;

        default:
            break;
        }

    }


}
```

大乔类也是同样的道理：

```
package bean;

/**
 * 大乔类
 * @author Administrator
 *
 */
public class Daqiao extends Hero{

    /**
     * 放大招的方法
     */
    public void DaZhao(Luban luban){
        luban.setAtHome(true);
    }

}
```

这个时候，我们可以看到，继承的一个好处就是可以省去很多重复的代码，提高了代码的复用率。

封装

现在我们考虑如何让java程序来播放一个音乐？

经过改造后的鲁班 kill 方法：

```
public void kill(){

    if(this.isAtHome){
        setName("鲁班七号");
        System.out.println(name + ":我是谁，在干什么？？");
        return;
    }

    killCount++;

    switch (killCount) {
    case 1:

        //解析音乐地址
        try{
            URL codebase = new URL("file:/E:/workspace/demos/1.wav");
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }
```

```
        System.out.println("First Blood!");
        break;

    case 2:

        //解析音乐地址
        try{
            URL codebase = new URL("file:/E:/workspace/demos/2.wav");
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }
        System.out.println("Double Kill!");
        break;
    case 3:

        //解析音乐地址
        try{
            URL codebase = new URL("file:/E:/workspace/demos/3.wav");
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }

        System.out.println("Triple kill!");
        break;
    case 4:
        //解析音乐地址
        try{
            URL codebase = new URL("file:/E:/workspace/demos/4.wav");
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }
        System.out.println("Quadra kill!");
        break;
    case 5:
        //解析音乐地址
        try{
            URL codebase = new URL("file:/E:/workspace/demos/5.wav");
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(2000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }
        System.out.println("Penta kill!");
        break;

    default:
        break;
    }

}
```

从代码中，我们可以发现，播放音乐的代码很多都是重复的，这个时候，我们就考虑能不能单独封装一个类，来播放音乐。

比如，我们可以新建一个音乐工具类，下次要播放音乐的时候，就调用这个类的方法。

```
package util;

import java.applet.Applet;
import java.applet.AudioClip;
import java.net.URL;

public class MusicUtil {

    /**
     *  音乐工具类
     * @param path 音乐地址
     * @param seconds 秒数
     */
    public static void playWav(String path,int seconds){
        try{
```

```
            URL codebase = new URL("file:/" + path);
            AudioClip a = Applet.newAudioClip(codebase);
            a.play(); //播放音乐
            Thread.sleep(seconds * 1000);
        }catch(Exception e){
            System.out.println("错就错呗，无所谓！");
        }
    }


    /**
     * @param args
     */
    public static void main(String[] args) {

        MusicUtil.playWav("E:\\workspace\\demos\\luban.wav", 5);

    }

}
```

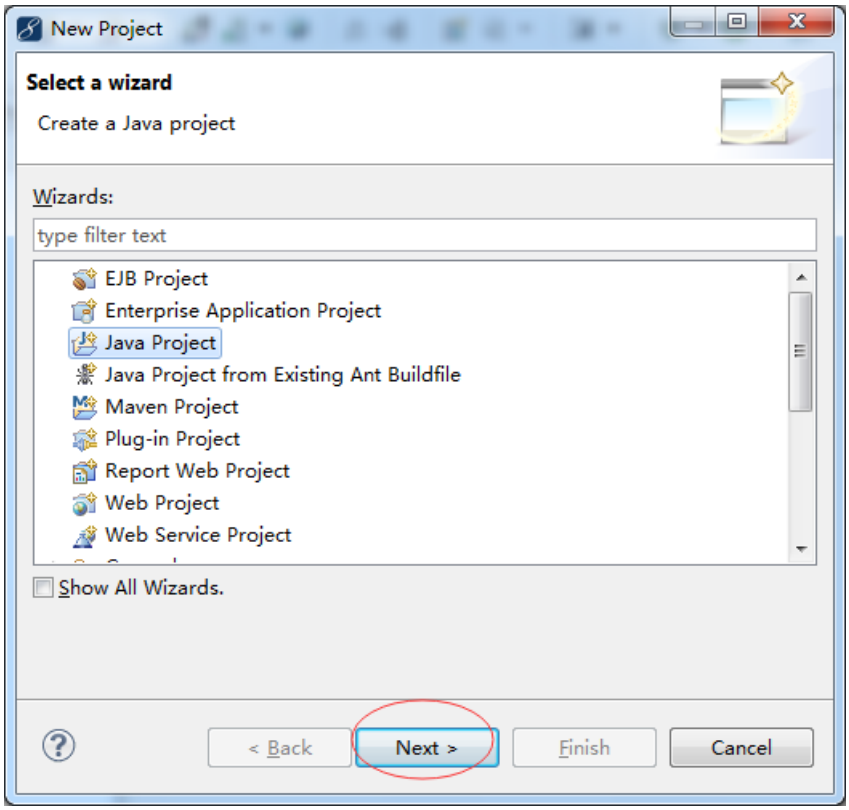
有了这个工具类，之前播放音乐的代码只需要一句话便可代替：

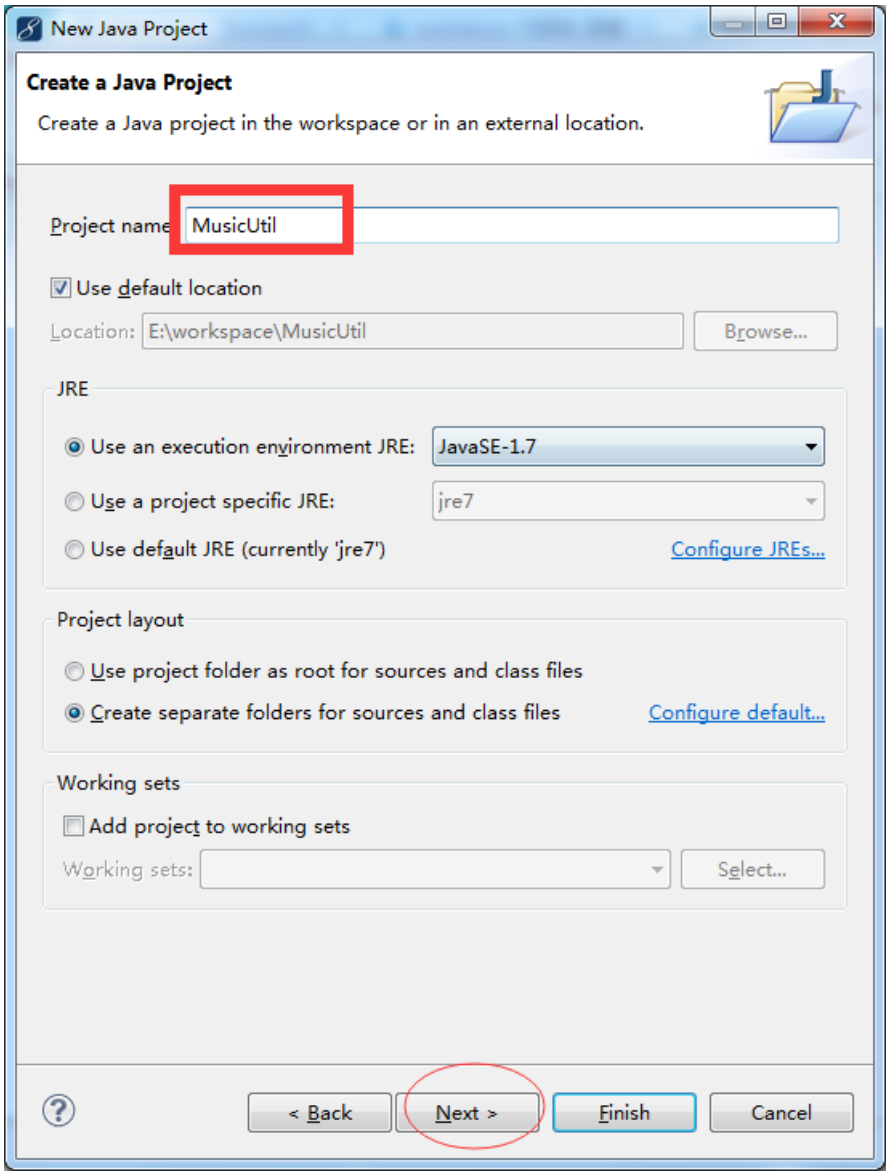
```
package bean;

import util.MusicUtil;

/**
 * 鲁班类
 * @author Administrator
 *
 */
public class Luban extends Hero{

    public void kill(){

        if(this.isAtHome){
            setName("鲁班七号");
            System.out.println(name + ":我是谁，在干什么？？");
            return;
        }

        killCount++;

        switch (killCount) {
        case 1:

            MusicUtil.playWav("E:/workspace/demos/1.wav", 2);

            System.out.println("First Blood!");
            break;

        case 2:

            MusicUtil.playWav("E:/workspace/demos/2.wav", 2);
            System.out.println("Double Kill!");
            break;
        case 3:

            MusicUtil.playWav("E:/workspace/demos/3.wav", 2);

            System.out.println("Triple kill!");
            break;
        case 4:
            MusicUtil.playWav("E:/workspace/demos/4.wav", 2);
            System.out.println("Quadra kill!");
            break;
        case 5:
            MusicUtil.playWav("E:/workspace/demos/5.wav", 2);
            System.out.println("Penta kill!");
            break;

        default:
            break;
        }

    }

    public void say(){

        MusicUtil.playWav("E:/workspace/demos/luban.wav", 5);
```

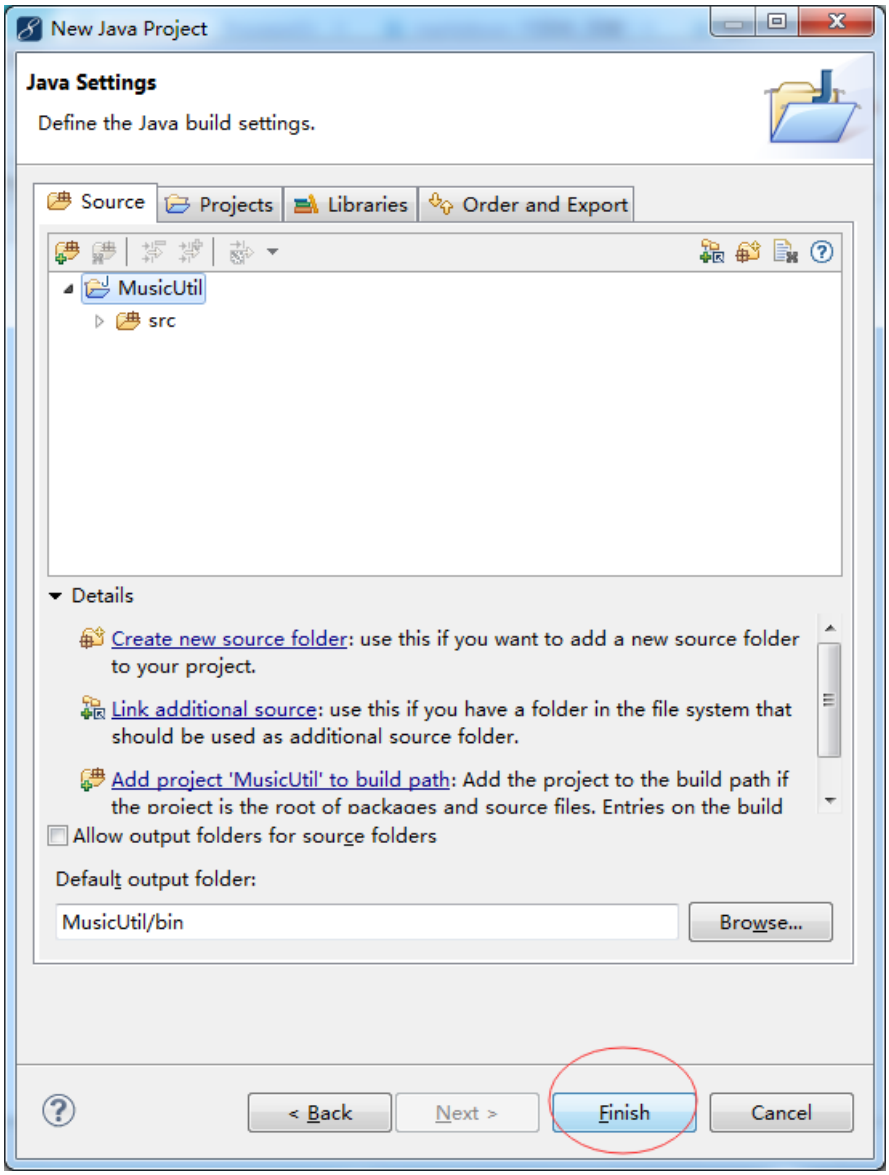
```
    }

}
```

甚至，这些工具类，我们还可以单独打成一个jar包，发布到网上，供别人使用！

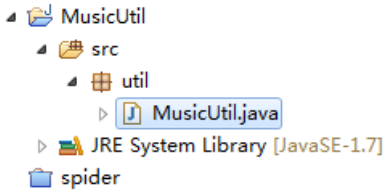
新建一个musicutil项目：



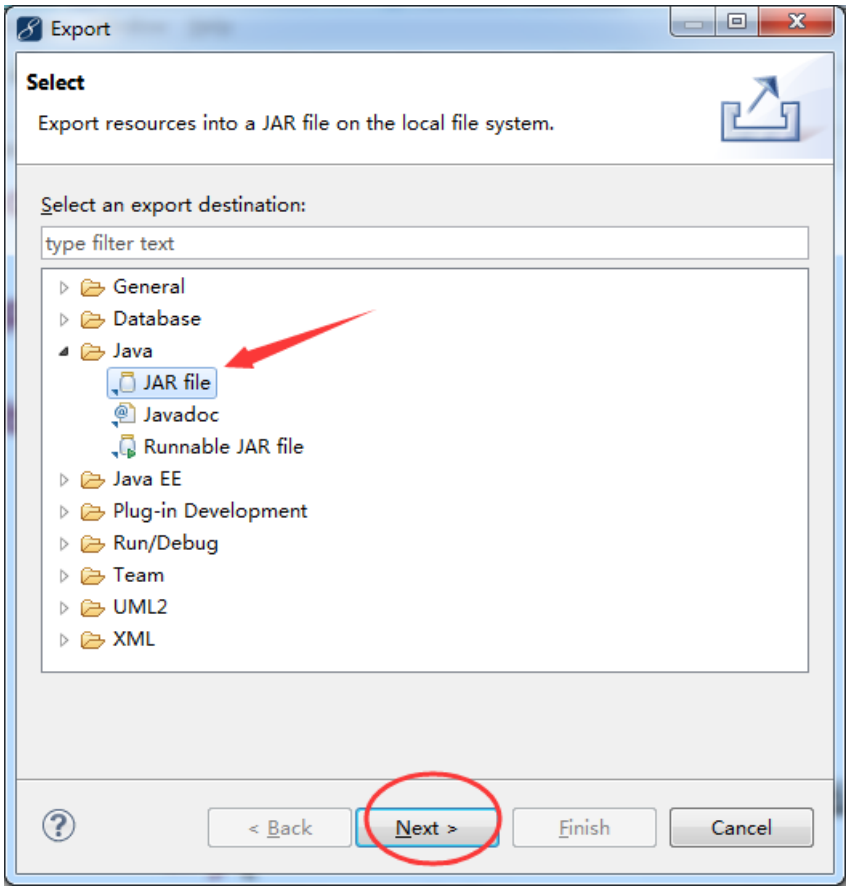

New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: MusicUtil
Use default location
Location: E:\workspace\MusicUtil
Browse...
JRE
Use an execution environment JRE: JavaSE-1.7
Use a project specific JRE: jre7
Use default JRE (currently 'jre7')
Configure JREs...
Project layout
Use project folder as root for sources and class files
Create separate folders for sources and class files
Configure default...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

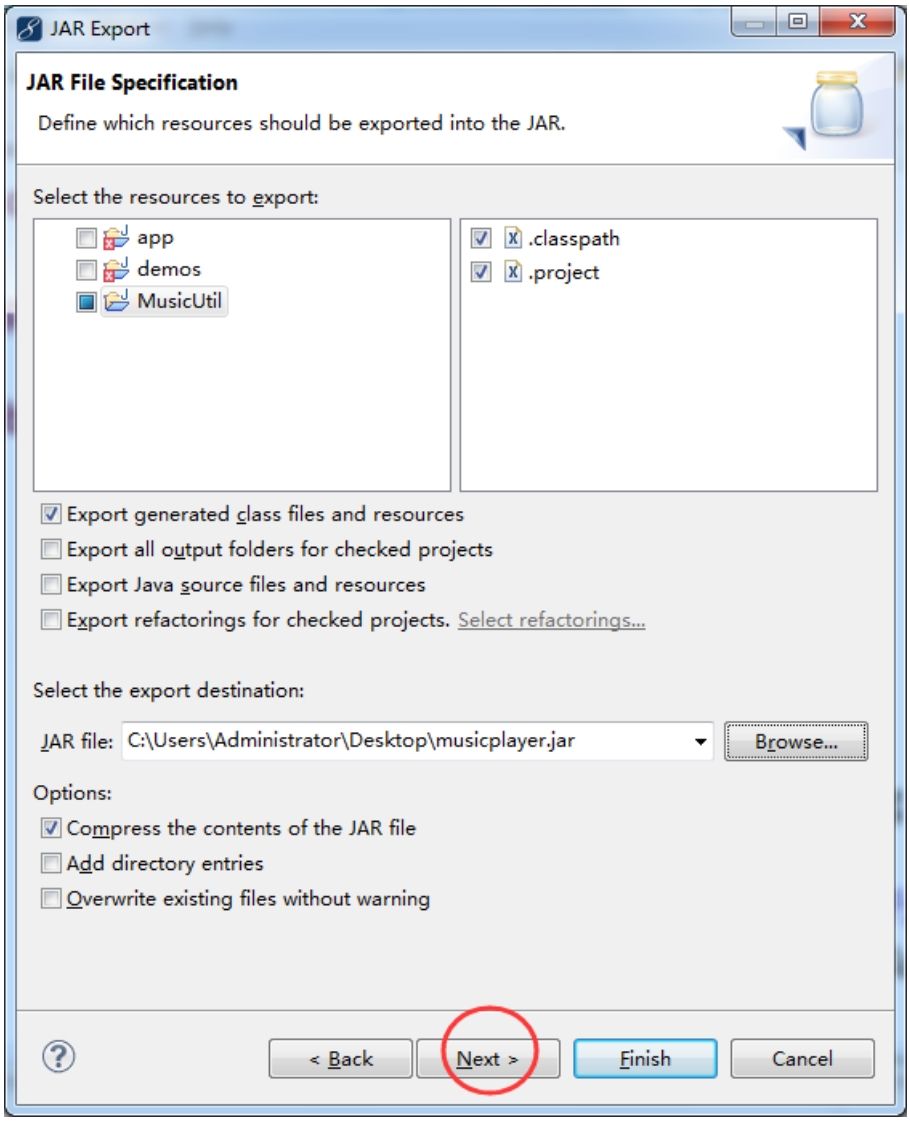


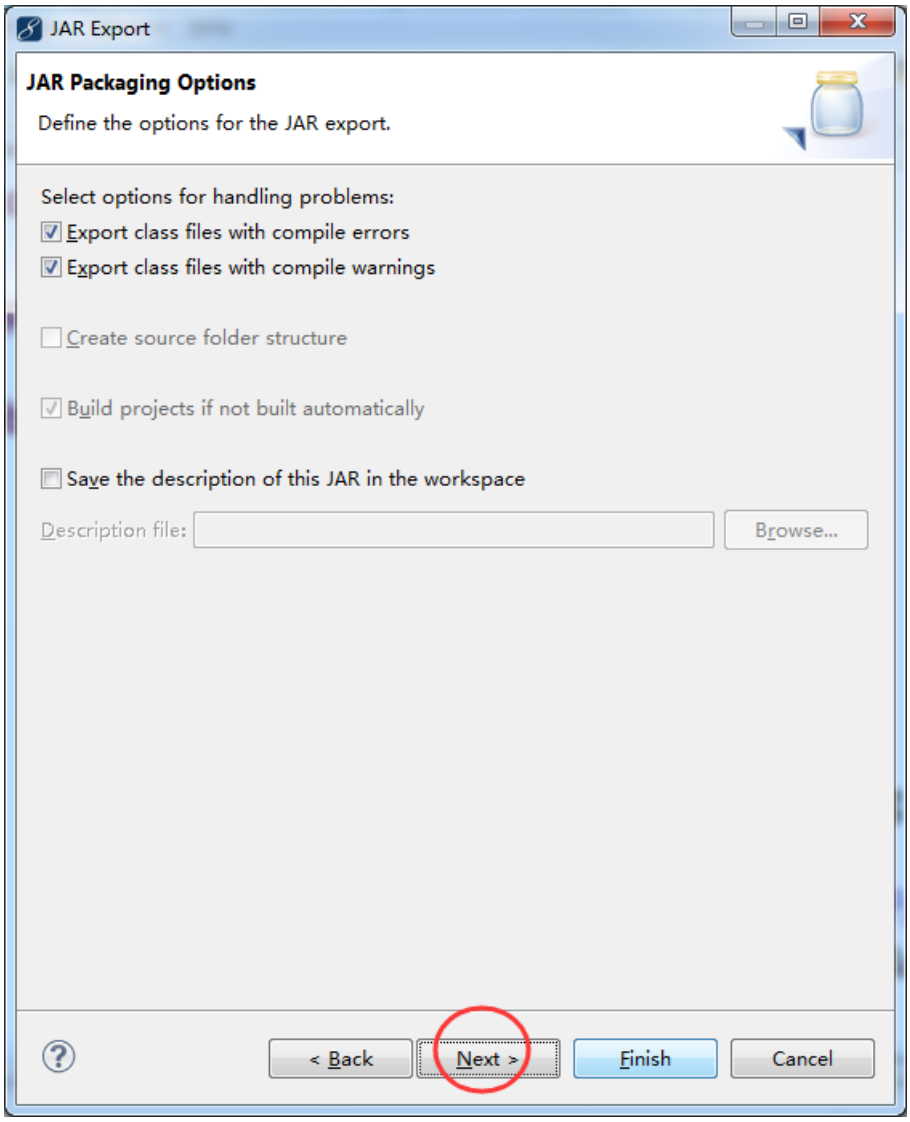
我们添加一个util包，把刚才的MusicUtil.java复制进来即可：

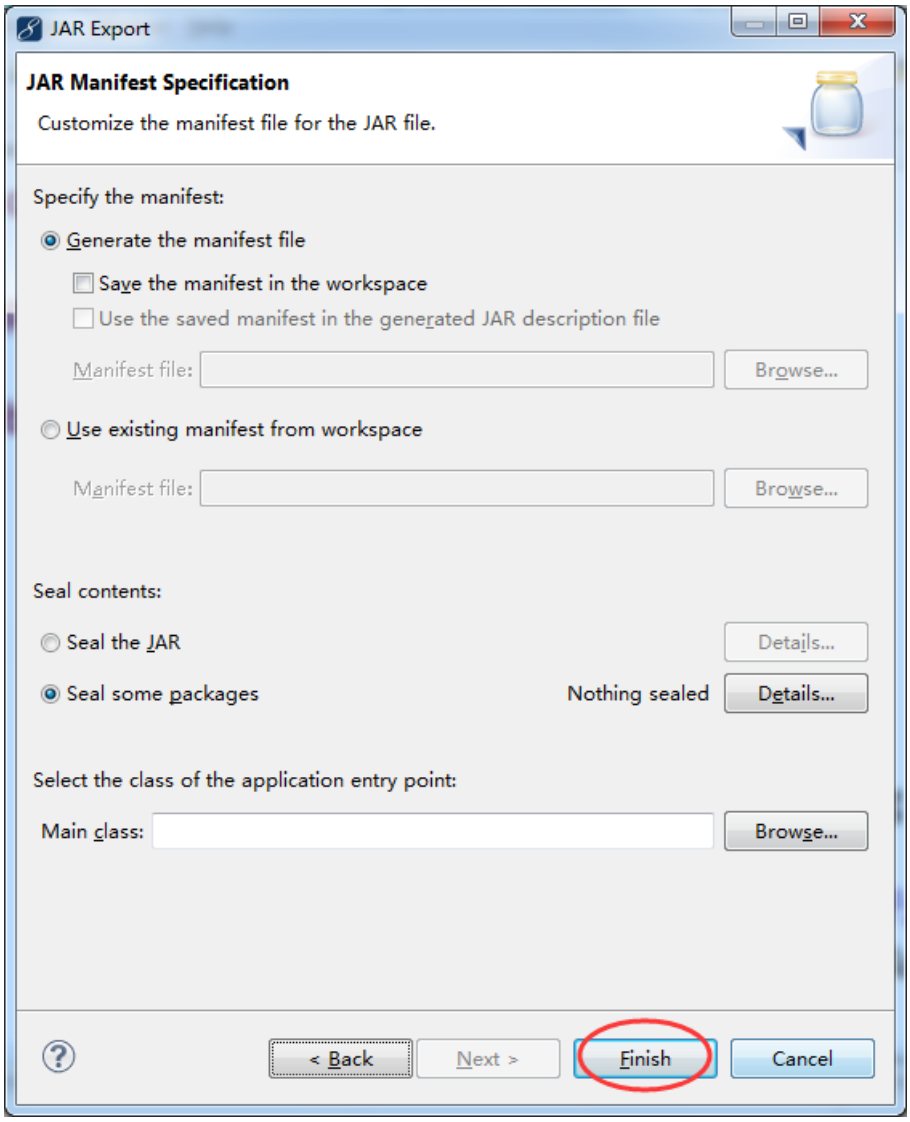


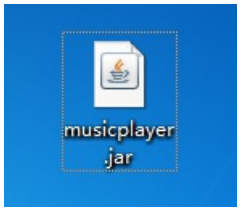
现在开始打包，右键项目，选择Export，Export：


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Database
Java
JAR file
Javadoc
Runnable JAR file
Java EE
Plug-in Development
Run/Debug
Team
UML2
XML
< Back
Next >
Finish
Cancel


JAR Export
JAR File Specification
Define which resources should be exported into the JAR.
Select the resources to export:
app
demos
MusicUtil
.classpath
.project
Export generated class files and resources
Export all output folders for checked projects
Export Java source files and resources
Export refactorings for checked projects. Select refactorings...
Select the export destination:
JAR file: C:\Users\Administrator\Desktop\musicplayer.jar
Browse...
Options:
Compress the contents of the JAR file
Add directory entries
Overwrite existing files without warning
< Back
Next >
Finish
Cancel


JAR Export
JAR Packaging Options
Define the options for the JAR export.
Select options for handling problems:
Export class files with compile errors
Export class files with compile warnings
Create source folder structure
Build projects if not built automatically
Save the description of this JAR in the workspace
Description file:
Browse...
< Back
Next >
Finish
Cancel





如果下次我们想要在其他项目中播放音乐的话，只需要引入这个jar即可，甚至，你要是闲得无聊的话，可以发布到网上，供别人下载使用。

回到封装的解说，我们可以把那些经常使用的，重复率特别高的代码，封装成一个方法，达到代码复用的目的。如果不封装，可想而知，我们的类中底层代码会特别多，不利于旁人理解，也不利于我们今后维护代码，因为方法名是我们自己取的，所以日后也可以很好的理解当初的代码是什么意思。

封装的另一个好处就是，有些东西，我不会，但是别人封装好了，我是不是可以去下载拿来用？？

又比如jdk给我们封装大部分java开发需要的类，这些类，很多都极其复杂，但是因为sun公司已经给我们封装好了，所以我们也完全不需要去关心其底层的实现，比如String，比如HashMap，直接拿来就用，而且效率很高！

所以，我认为，封装另一个好处就是，我不会的东西，只要有人会，就好了，我照样可以拿来开发程序！

多态

```
/**
 * 大乔类
 * @author Administrator
 *
 */
public class Daqiao extends Hero{

    /**
```

```
     * 放大招的方法
     */
    public void DaZhao(Luban luban){
        luban.setAtHome(true);
        luban.say();
    }

}
```

回顾大乔类，问一下大家，大乔的大招对鲁班有效，对兰陵王有没有效？对露娜有没有效？

如果我们现在新建了一个露娜类：

```
package bean;

import util.MusicUtil;

/**
 * 露娜类
 * @author Administrator
 *
 */
public class Luban extends Hero{

    public void kill(){

        if(this.isAtHome){
            setName("露娜");
            System.out.println(name + ":我是谁，在干什么？？");
            return;
        }

        killCount++;

        switch (killCount) {
        case 1:

            MusicUtil.playWav("E:/workspace/demos/1.wav", 2);

            System.out.println("First Blood!");
            break;

        case 2:

            MusicUtil.playWav("E:/workspace/demos/2.wav", 2);
            System.out.println("Double Kill!");
            break;
        case 3:

            MusicUtil.playWav("E:/workspace/demos/3.wav", 2);

            System.out.println("Triple kill!");
            break;
        case 4:
            MusicUtil.playWav("E:/workspace/demos/4.wav", 2);
            System.out.println("Quadra kill!");
            break;
        case 5:
            MusicUtil.playWav("E:/workspace/demos/5.wav", 2);
            System.out.println("Penta kill!");
            break;

        default:
            break;
        }

    }


    public void say(){

        MusicUtil.playWav("E:/workspace/demos/zixia.wav", 5);
    }


}
```

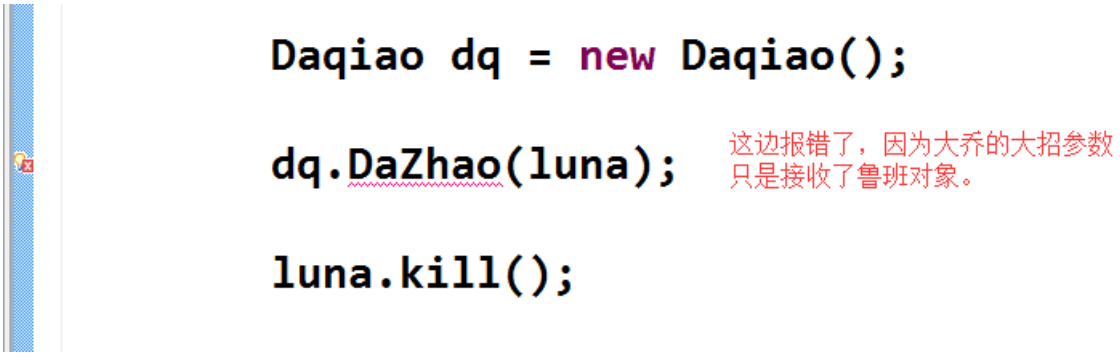
同时，在父类中，添加say方法：

```
public void say(){
```

```
}
```

这个时候，大乔的大招是无法装进露娜类的。

测试：



对大乔来说，不管你是什么英雄，只要你继承了Hero类，都可以分分钟送你回家！所以，我们要改变大招方法的参数：

```
/**
 * 放大招的方法
 */
public void DaZhao(Hero hero){
    hero.setAtHome(true);
    hero.say();
}
```

这就是所谓的多态。

从这个例子当中，我们可以看出，多态最直接的好处就是某个方法中，同一个参数可以接收多种类型的数据。

比如刚才大乔的例子，如果我们不用多态，那么大招方法就得重载很多个，比如针对鲁班，就得有一个大招方法。针对兰陵王，又需要一个大招方法。这样会导致方法特别多，而且不易维护。如果，我们直接设定所有的英雄都必须继承自英雄类，那么，大招方法，只需要一个英雄类的参数即可，一个方法足以。

所以，多态的好处，我认为，就是方便传参。而且，更多的，在设计类的时候，就应该预想到会有什么类型的参数传进来，这是需要提前考虑的。

作业：
用面向对象的思维来描述以下的现象：

兰陵王被王昭君的大招冻住了，结果对方的项羽把兰陵王推了出去！

提示：
兰陵王，王昭君，项羽 都应该继承自Hero类。
