《软件工程实验》

实验指导书

1 实验目标

1）使学生加深对软件工程基本概念的理解，学会使用软件工程的思想、方法指导软件开发。

2）通过实验，要求学生掌握在软件生存周期各阶段使用到的软件开发工具，能独立安装并使用各类工具。

3）培养学生利用软件工具进行系统分析、设计、实现等实际能力。

2 实验项目及学时分配

序号	实 验 项 目 名 称	实验学时	每组人数	实验类别	实验类型
1	**结对编程**	4	2	专业基础	验证与设计
2	**单元测试**	4	2	专业基础	验证与设计
3	**代码评审与程序性能优化**	4	2	专业基础	验证与设计
4	**UML 建模工具的安装与使用**	4	1	专业基础	验证与设计
5	**UML 系统分析与设计**	8	1	专业基础	综合
6	**系统测试**	4	1	专业基础	验证与设计
7	**Git 实战**	4	1	专业基础	验证与设计

3 单项实验内容及要求

实验要求学生按照面向对象方法，在软件工程思想指导下，独立或分组完成各项实验，每个实验结束均需撰写和提交本实验的实验报告。

项目名称	实验内容
实验一：结对编程	练习结对编程(pair programming)，体验敏捷开发中的两人合作；两人一组，自由组合；使用一台计算机，共同编码，完成实验要求。
实验二：单元测试	安装 JUnit 单元测试工具；使用 Junit 对实验程序进行单元测试。
实验三： 代码评审与程序性能优化	针对前面实验中所完成的代码，进行代码评审(走查)和性能分析；练习代码评审的两个方面：静态分析、动态分析。

实验四： **UML 建模工具的安装与使用**	安装并学习使用一种 UML 建模工具；了解该工具图形应用的基本概念，重点掌握用例图、顺序图、状态图等 UML 模型图的绘制方法。
实验五： **UML 系统分析与设计**	选定一个系统案例，使用上一个实验中所安装的 UML 建模工具完成系统的分析与设计，绘制相应的 UML 模型图。
实验六：系统测试	在初步掌握 JMeter 系统测试工具的使用方法之后，通过对一个软件实例来组织测试，以进一步熟悉 JMeter；通过分析和采用适当的测试用例去发现程序中的错误，提高软件测试的实践能力。
实验七：Git 实战	学习软件配置管理工具 Git 的相关内容；我们知道版本控制，对于大中型软件系统的开发；利用假想的 IT 项目，介绍该软件的使用方法。

4 实验组织

所有实验均以个人独立完成，或两人结对协作的方式展开，在实验室环境下进行。实验指导教师帮助学生熟悉各类软件工具，培养符合软件工程方法的软件开发技能；督促学生在正式实验前进行适当的预习或准备，在实验过程中按照实验步骤积极动手进行实验操作，深入思考、分析和讨论，在课堂外再进行一定时间的练习，按各个实验的具体要求完成和提交实验成果，增强解决实际问题的能力。

5 考核方式

结合平时考查，以实验实际操作的优劣、提交的文档和程序等作为考核依据。实验成绩按五级制评分，课程结束，以学生实际实验工作能力的强弱作为评定成绩的主要依据。

最终成绩 100% = 平时成绩 10% + 实验成绩 70% + 课程总结（含案例）20%

6 推荐教材及参考资料

1）《软件工程基础实验指导书》 自编

2）《构建之法：现代软件工程》，邹欣，人民邮电出版社，2015

3）《UML 与 Enterprise Architect 7.5 团队开发实务手册》，赖信仁著，UMLChina 改编，电子工业出版社，2010

4）《零成本实现 Web 性能测试：基于 Apache JMeter 》温素剑编，电子工业出版社，2012

实验一 结对编程

一、实验目的

1）体验敏捷开发中的两人合作。

2）进一步提高个人编程技巧与实践。

二、实验内容及要求

1）选择一个程序实例，练习结对编程(pair programming)实践；

2）要求学生两人一组，自由组合。每组使用一台计算机，二人共同编码，完成实验要求。

3）要求在结对编程工作期间，两人的角色至少切换 4 次；

4）编程语言不限，版本不限。建议使用 Python 或 JAVA 进行编程。

三、示例程序问题描述

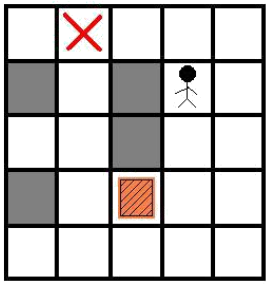
（1）生命游戏

生命游戏是英国数学家约翰·何顿·康威在 1970 年发明的细胞自动机，它包括一个二维矩形世界，这个世界中的每个方格居住着一个活着的或死亡的细胞。一个细胞在下一个时刻生死取决于相邻八个方格中活着的或死了的细胞的数量。如果相邻方格活着的细胞数量过多，这个细胞会因为资源匮乏而在下一个时刻死去；相反，如果周围活细胞过少，这个细胞会因太孤单而死去。

游戏在一个类似于围棋棋盘一样的，可以无限延伸的二维方格网中进行。例如，设想每个方格中都可放置一个生命细胞，生命细胞只有两种状态：“生”或“死”。图中，用黑色的方格表示该细胞为“死”, 其它颜色表示该细胞为“生” 。游戏开始时, 每个细胞可以随机地（或给定地）被设定为“生”或“死”之一的某个状态, 然后，再根据如下生存定律计算下一代每个细胞的状态：

- 每个细胞的状态由该细胞及周围 8 个细胞上一次的状态所决定；
- 如果一个细胞周围有 3 个细胞为生，则该细胞为生，即该细胞若原先为死则转为生，若原先为生则保持不变；
- 如果一个细胞周围有 2 个细胞为生，则该细胞的生死状态保持不变；
- 在其它情况下，该细胞为死，即该细胞若原先为生则转为死，若原先为死则保持不变。

（2）推箱子游戏



在一个 M*N 的房间里有一个箱子和一个搬运工，搬运工的工作就是把箱子推到指定的位置。注意：搬运工只能推，不能拉。因此，如右图所示，箱子被推到一个角上，那么就不能再被移动了，如果箱子被推到一面墙上，那么箱子只能沿着墙移动。

要求：现在给定房间的结构、箱子的位置、搬运工的位置和箱子要被推去的位置，请计算出搬运工至少要推动箱子多少格？

输入：输入数据的第一行是一个整数 T(1<=T<=20)，代表测试数据的数量。然后是 T 组测试数据，每组测试数据的第一行是两个正整数 M，N(2<=M，N<=7)，代表房间的大小。然后是一个 M 行 N 列的矩阵，代表房间的布局，其中 0 代表空的地板，1 代表墙，2 代表箱子的起始位置，3 代表箱子要被推去的位置，4 代表搬运工的起始位置。

输出：对于每组测试数据，输出搬运工最少需要推动箱子多少格才能帮箱子推到指定位置，如果不能推到指定位置则输出-1。

实验二 单元测试

一、实验目的

1）掌握单元测试的方法；

2）学习 JUnit 测试原理及框架；

3)掌握在 Eclipse 环境中加载 JUnit 及 JUnit 测试方法和过程。

二、实验内容与步骤

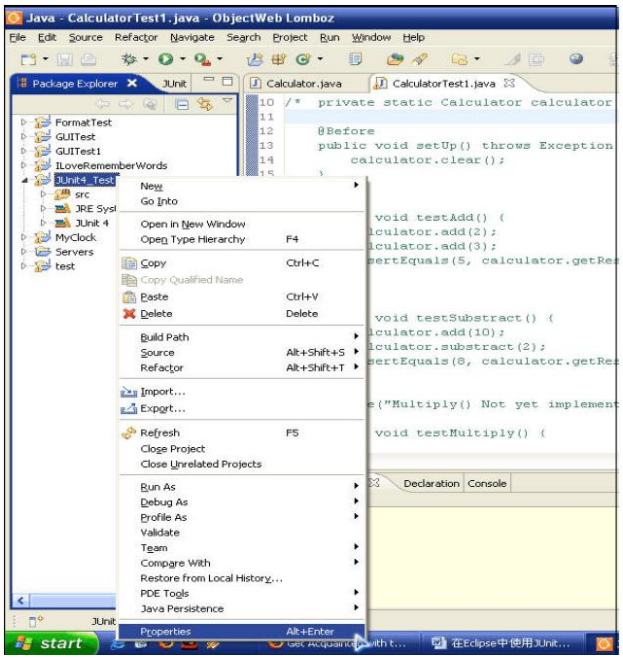
1. Eclipse 中 JUnit 的使用

Eclipse 集成了 JUnit，可以非常方便地编写 Test Case。Eclipse 自带了一个 JUnit 插件，不用安装就可以在项目中测试相关的类，并且可以调试测试用例和被测类。下面以实例说明，如何建立一个基于 JUnit4 的测试项目，对一个类当中的多个方法进行单元测试。

（1）新建一个名为 JUnitTest 的项目，在其中编写一个 Calculator 类，这是一个能够简单实现加减乘除、平方、开方的计算器类，然后对这些功能进行单元测试。这个类中我们故意保留了一些 Bug 用于演示，这些 Bug 在注释中都有说明。该类代码如下：

```
public class Calculator{
    private static int result;          // 静态变量，用于存储运行结果
    public void add(int n) {     result = result + n;   }
    public void substract(int n) {
          result = result - 1;        //故意的 Bug，应该是  result =result-n
     }
    public void multiply(int n) {   }   // 假设此方法在项目完成过程中尚未写好
    public void divide(int n) {   result = result / n;   }
    public void square(int n) {     result = n * n;   }
    public void squareRoot(int n) { //求平方根
          for (; )       //Bug : 死循环
     }
    public void clear() {   // 将结果清零
          result = 0;
     }
}
```

（2）将 JUnit4 单元测试包引入这个项目：在该项目上点右键，点“属性”，如右图所示：



（3）在弹出的属性窗口中，首先在左边选择“Java Build Path”，然后到右上选择“Libraries”标签，之后在最右边点击“Add Library…”按钮，如下图所示：

然后在新弹出的对话框中选择 JUnit4 并点击确定，如上图所示，JUnit4 软件包就被包含进我们这个项目了。

（3）生成 JUnit 测试框架：在 Eclipse 的 Package Explorer 中用右键点击该类弹出菜单，选择“JUnit 测试用例”。在弹出的对话框中，进行相应的选择，如下图所示：

点击“下一步”后，系统会自动列出你这个类中包含的方法，选择你要进行测试的方法。此例中，我们仅对“加、减、乘、除”四个方法进行测试。如右图所示：

之后系统会自动生成一个新类 CalculatorTest，里面包含一些空的测试用例。你只需要将这些测试用例稍作修改即可使用。

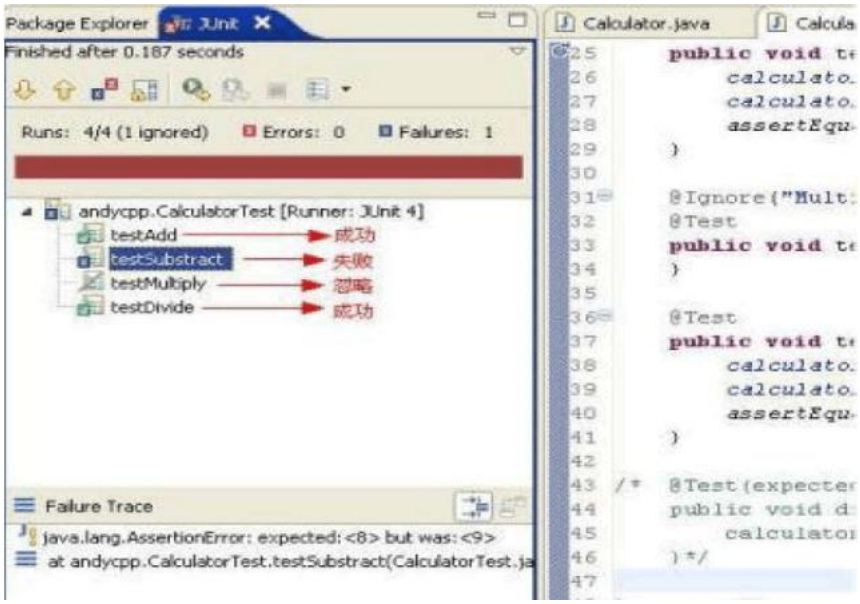
完整代码如下：

```
package andycpp;
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import org.junit.internal.runners.TestClassRunner;
import org.junit.runner.RunWith;

@RunWith(TestClassRunner.class)
public class CalculatorTest {
	private static Calculator calculator=new Calculator();
	@Before
	public void setUp() throws Exception {
		calculator.clear();
	}
	@After
	public void tearDown() throws Exception {
	}
	@Test(timeout=1000)
	public void testAdd() {
		calculator.add(2);
		calculator.add(3);
		assertEquals(5, calculator.getResult());
	}
	@Test
	public void testSubstract() {
		calculator.add(10);
		calculator.substract(2);
		assertEquals(8,calculator.getResult());
	}
	@Ignore("Multiply() Not yet implemented")
	@Test
	public void testMultiply() {
	}
	@Test(expected =ArithmeticException.class)
	public void testDivide() {
		calculator.add(8);
		calculator.divide(0);
		assertEquals(4,calculator.getResult());
	}
}
```

（4）运行测试代码：按照上述代码修改完毕后，我们在 CalculatorTest 类上点右键，选择

“Run As ——>JUnit Test”来运行我们的测试，运行结果如下：



进度条是红颜色表示发现错误，具体的测试结果在进度条上面有表示“共进行了 4 个测试，其中 1 个测试被忽略，一个测试失败”。

2. 利用 JUnit 对“实验一”中的各个类，进行单元测试。

实验三 代码评审与程序性能优化

一、实验目的

1）了解代码审查的含义；

2）了解如何对程序进行性能优化；

3）掌握配置工具的安装与使用；

二、实验内容及要求

1) 针对前面“实验一”中所完成的代码，进行代码评审(走查)和性能分析，从时间性能角度对代码进行优化；

2) 练习代码评审的两个方面：静态分析、动态分析；

3) 使用以下三个工具完成实验：

- Checkstyle
- FindBugs
- PMD

4) 按“实验一”的分组方式，两人一组，随机分配另一组的代码作为本组评审和分析的对象，实验期间不能与原作者进行沟通。

三、实验过程

（1）在 Eclipse 中配置代码审查与分析工具。要求学生采用屏幕截图的方式给出在 Eclipse 中配置 Checkstyle、PMD 和 Findbugs 的过程。

（2）分别使用这些工具对原始代码进行评审和性能分析，记录结果，期间不要有任何修改。

（3）对工具执行结果进行人工分析，对三种工具的分析结果进行对比，找到它们发现问题的能力差异。

（4）根据结果对源代码进行修正（代码规范、性能）；

（5）重新使用工具进行评审和性能分析，直到无法再改进为止。

实验四 UML 建模工具的安装与使用

一、实验目的

1）学习使用 EA（Enterprise Architect）开发环境创建模型的一般方法；

2）理解 EA 界面布局和元素操作的一般技巧；

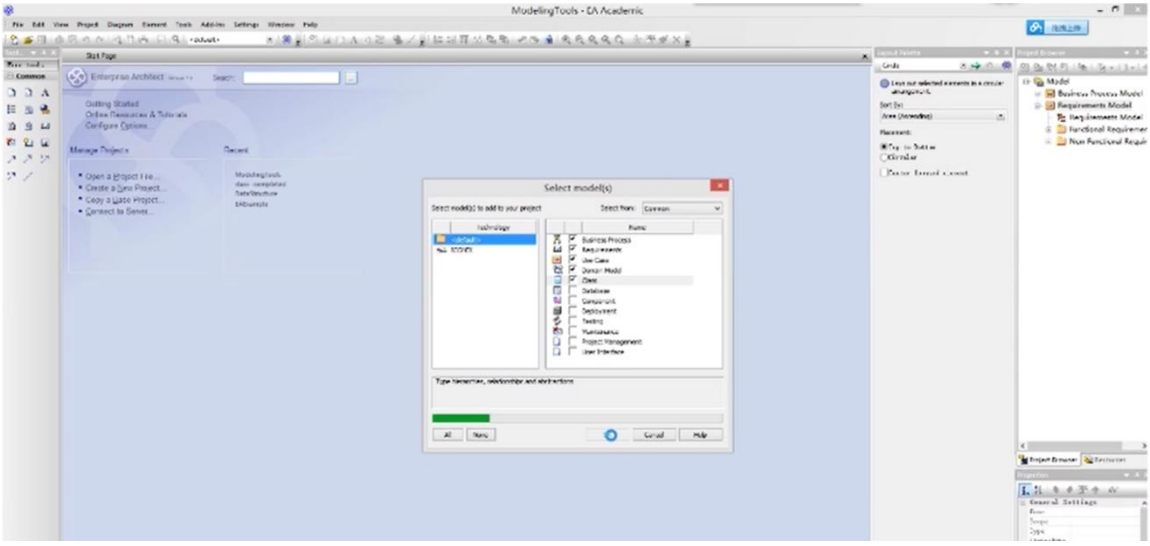
3）熟悉 UML 中的各种图的建立和表示方法；

4）掌握如何通过 EA 工具完成相关模型的建立。

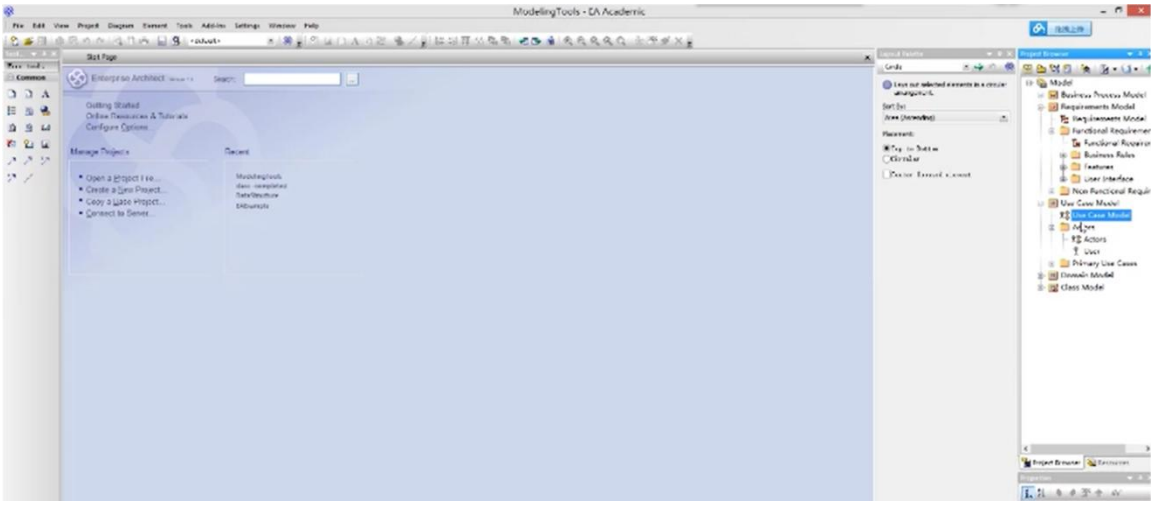
二、实验内容及步骤

1. EA 开发环境的介绍

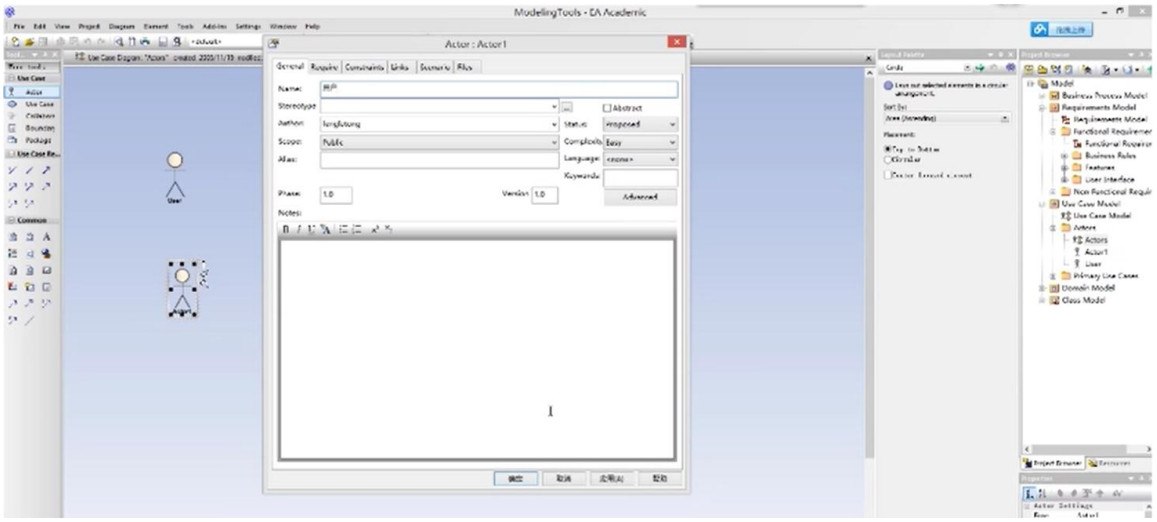
（1） 开始——>运行——>Enterprise Architect，打开 Enterprise Architect 软件。进入软件之后，可以选择新建一个项目。在新建项目时，系统会提示选择所需要的模型设计。



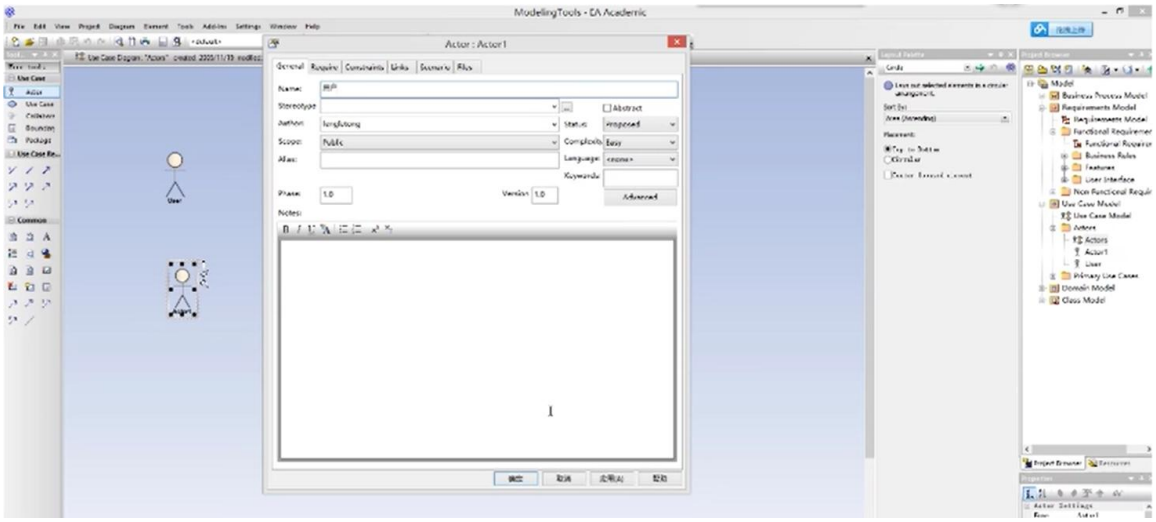
（2）选择了所需要的模型之后，可以看到，在界面的右侧出现了相应的导航栏。如下图所示，在导航栏里面列出了刚才所选择的系统模型。



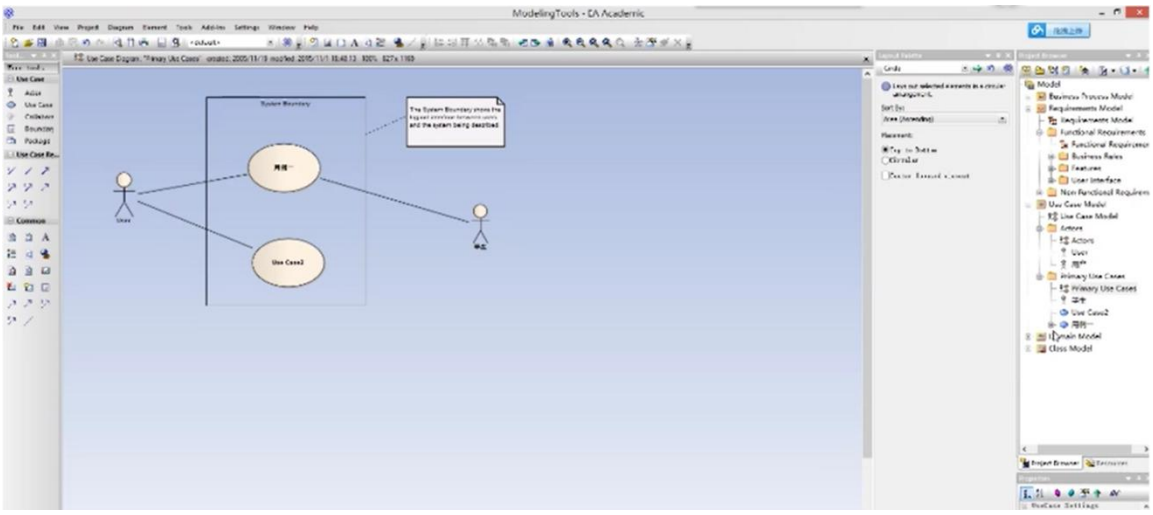
（3）假设选择用例建模，那么可以从左侧的工具面板中拖拽出一个参与者，并为它命名。这样在系统里面就多了一个叫做“用户”的参与者。



（4）以同样的方式从左侧工具面板中拖曳一个用例，命名为“用例一”。



（5）同样通过拖拽的方式，建立用例与参与者之间的关系。对这个关联关系，我们还可以设置更加细化的约束。



2. 学习和掌握如何通过 EA 工具完成示例系统的各类 UML 模型的建立。

实验五 UML 系统分析与设计

一、实验目的

1）理解软件开发过程的几个基本阶段；

2）掌握基本的软件分析与设计技术；

3）熟练运用 EA 工具进行系统建模。

二、实验内容与要求

要求学生以微信抢票系统为例，使用 EA 工具完成该系统模型的创建，绘制相应的 UML 模型图。要求：

（1）为系统创建用例模型，使用 EA 工具完成用例图的绘制，并给出相应的用例描述；

（2）使用 EA 工具绘制系统的类图和包图，给出类的描述以及类与类之间的关系；

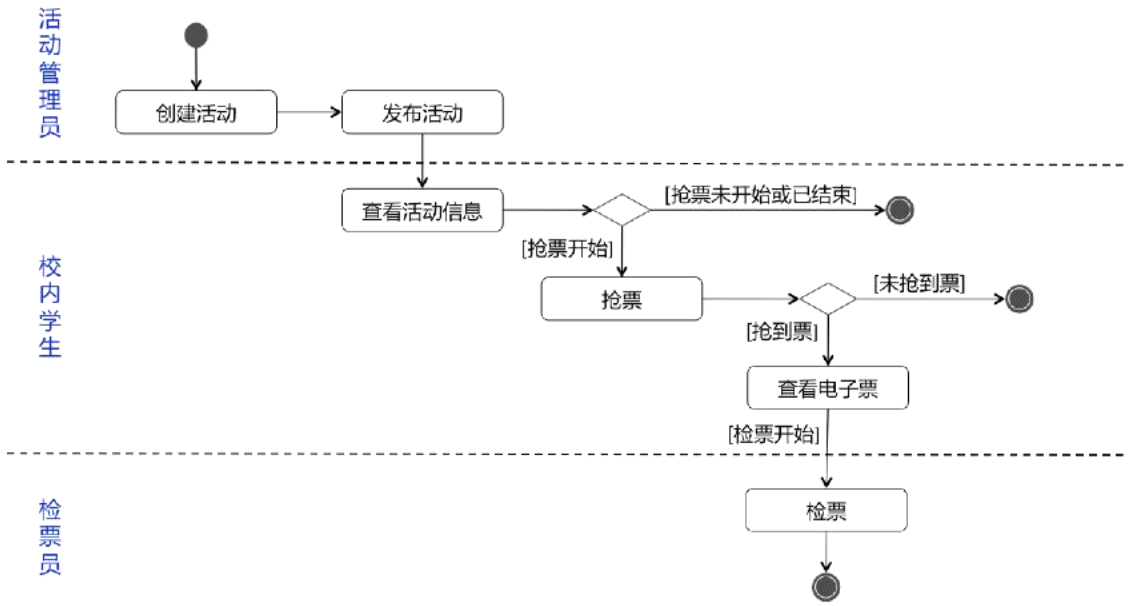
（3）为系统进行行为建模，使用 EA 工具绘制系统的顺序图、活动图、交互图等。

【示例系统】微信抢票系统的问题背景与系统需求如下：

某学校团委经常组织一些活动，让组织人员非常头疼的一个问题是活动票的发放。尤其是一些受学生欢迎的热门活动，因为原来采取排队领票或购票形式，经常出现的场景就是在活动票发放前两三个小时门外就排起长长的队伍。为了改善学生排长队领票的不便，校团委希望开发一款微信抢票应用，为学生们提供热门活动推送、校园活动抢票等服务。

整个抢票过程包括创建和发布活动、抢票和检票三个阶段，具体过程如下图所示。校团委相关部门负责创建和发布活动，校内学生可以查看活动的详细信息。在抢票开始时，学生可以参与指定活动的抢票。如果学生成功抢到活动票，系统将自动生成一个带有二维码的电子票，学生可以查看自己的电子票；如果没有抢到票，系统将未抢到票的消息反馈给学生。

在活动开始时，校团委工作人员在活动入场处进行检票，学生可以持电子票检票参加活动。对于学生持有的电子票，工作人员使用二维码扫描枪进行扫描，验票成功即可入场，验票成功的条件是电子票有效且未被使用；学生也可以持自己的学生证，由工作人员通过学号查询电子票，再手动确认检票。



根据上述业务流程和要求，确定了微信抢票应用的系统需求。

（1）功能需求

- 活动管理员可以发布和维护最新的校园活动信息，包括活动名称、活动详情介绍、活动时间、活动地点、抢票数量、抢票时间等。
- 本校学生使用自己在学校信息门户的账号和密码实现微信号与校园账号的绑定。
- 学生可以查看校园活动的详细信息。
- 学生可以在活动抢票时段进行微信抢票，目前规定一个账号一次只能抢一张票。
- 学生在抢票成功之后可以获得系统生成的一张二维码电子票。
- 抢到票的学生在抢票未结束时可以退票。
- 抢到票的学生在活动开始时可以使用电子票通过检票进入活动现场。

2. 非功能需求

- 系统应能够支持 500 个用户并发访问。
- 系统应当支持 iOS 和 Andriod 两种主流手机操作系统。
- 在正常网络环境下，系统的响应速度应该控制在 5 秒以内。
- 所交付的系统源代码要求格式规范、风格统一，易于阅读和维护。
- 系统应该具有良好的架构设计，可扩展性强。
- 系统应具有良好的用户体验，并充分体现微信的交互特点。
- 系统应该保证安全可靠。

实验六 系统测试

一、实验目的

（1）了解负载测试、压力测试等性能测试的概念。

（2）能使用常用工具 JMeter 进行性能测试并对根据测试结果进行性能分析。

（3）进一步掌握软件压力测试的常用方法。

二、实验内容

根据后面给出的关于 JMeter 的使用介绍，打开 JMeter 测试软件，了解其功能结构，并选择对一个已有的网站或自己事先设计好的动态或静态网站（页面），进行测试和分析。

1. JMeter 结合 JUnit 实验

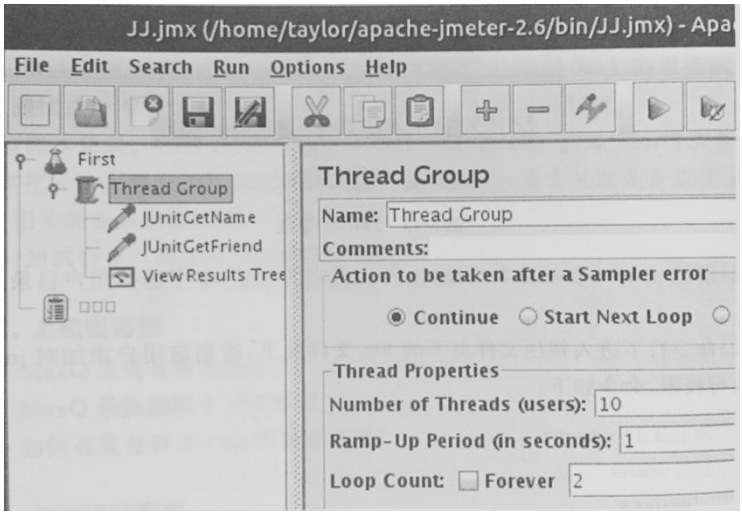
（1）练习 Eclipse 编写 Java 程序，先打开 Eclipse，编写一个新的类 com.opensource.jmeter，要求该类包含一个 name(String)属性，一个 friend(String)属性以及 getName、setName、getFriend、setFriend 四个方法。其余额外的属性和方法自行处理。

（2）练习 JUnit 编写测试用例。编写一个 JUnit 测试用例 com.opensource.JUnit,要求至少测试 getName 和 getFriend 两个方法。

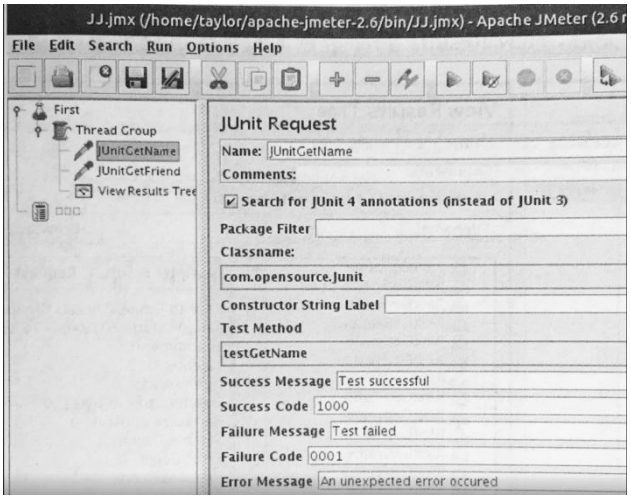
（3）打包。把上面的类和测试用例打包到“JUnit 安装目录/lib/junit”下的 test.jar 包里。

（4）开启 JMeter。点击左边窗口第一项，出现“Test Plan”的控制面板。在下面的 name 后面输入喜欢的名称，比如：“My First Test Plan”。再点击左边窗口第 2 项，第一项变成了刚才输入的名称。

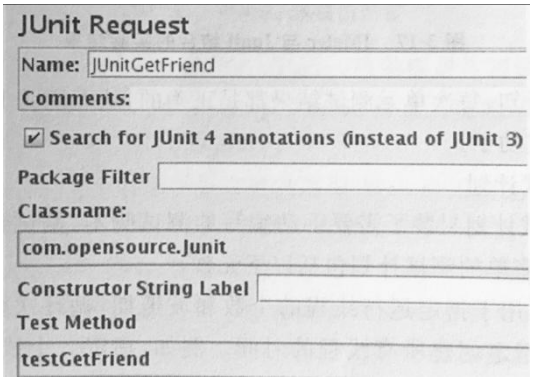
（5）编写测试案例。点击左边窗口第 1 项,在弹出菜单中选择 Add-Threads(Users)-Thread Group。点击左边窗口的 Thread Group，修改右边的“Number of Threads(Users)”为 10，“Loop Count”为 2。如下图所示，JMeter 用 thread 即线程来模拟一个用户，所以有多少个线程就相当于有多少个用户在同时操作，Loop Count 指的是每个线程的循环次数。



右击上图左边窗口中的 Thread Group，在弹出菜单中选择 Add-Sampler-JUnit Request，总共添加两个 JUnit Request，修改右边的 Name 为 JUnitGetName，勾选“Search for JUnit 4 annotations(instead of JUnit 3)”，Classname 选 com.opensource.JUnit，Test Method 选 testGetName，总体设置如下图所示：



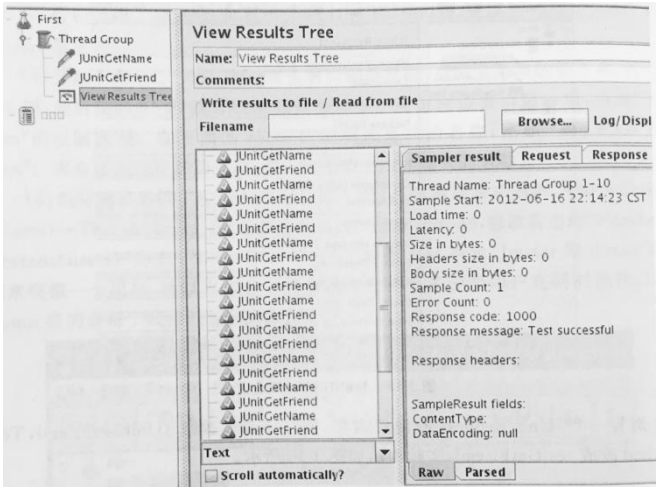
对另一个 JUnit Request 的设置类似，除了 Name 改为 JUnitGetFriend，Test Method 改为 testGetFriend，实验图示如下所示：



右击左边窗口中的 Thread Group，在弹出菜单中选择 Add-Listener-ViewResultsTree。

（6）运行。通过上述步骤模拟了 10 个用户，每个用户按顺序进行 testGetName 和 testGetFriend 两个单元测试 2 次，即每个用户执行 4 次测试，顺序是 Name，Friend，Name，Friend。

点击工具栏中“Start”按钮，会跳出需要保存 plan 的提示，保存后即开始运行 plan。等运行完后，View Results Tree 控制面板下会出现数据，如下图所示。



由图中数据可知，每次单元测试结果都是正确的，从而说明了 10 个用户 2 次循环的压力单元测试成功了。

2. JMeter 测试计划

JMeter 的测试计划封装了需要手动编写的测试脚本，测试计划定义了如何进行测试的框

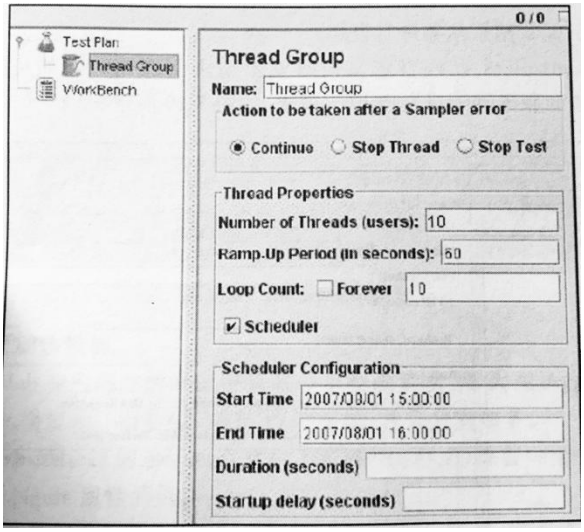
架，最简单的测试计划包括以下元素。

第一，线程组：用于指定运行线程的个数和坡道期，循环次数。每个线程模拟一个用户，坡道期指定创建所有线程的时间。例如，指定 5 个线程和 10 秒坡道期，表示每个线程的创建时间为 2 秒。循环次数定义了测试重复的次数，而可以指定开始和结束的具体时间。

第二，采样器：用于发出 http/ftp/soap/xml/jdbc/ldap 请求到服务器。

第三，监听器：用于后处理请求数据。例如，可以保存数据到文件或者以图表显示。

（1）线程组。测试计划至少由一个线程组组成。线程组是测试计划的入口，模拟多用户请求，多个线程组可彼此独立执行。每一线程组中包含采样器、逻辑控制器、配置、监听器和定时器的一个或多个组合。每个采样器能够关联一个或多个预处理元素、后处理元素和断言元素，如下图所示。



Action to be taken after a Sample error：一旦采样器在测试过程中报错，可作以下选择：Continue 继续进行下面的测试，Stop Thread 停止当前测试的线程，Stop Test 完成停止测试，便于进行错误原因的查看。

Number of Threads：模拟用户的人数或者是 Web 应用的连接数。

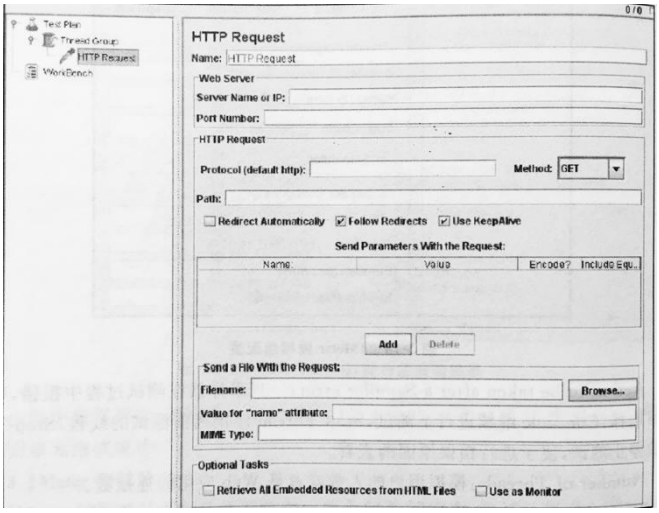
Ramp-Up Period：定义 JMeter 创建所有测试线程的时长。例如，设定线程数为 10，坡道期为 60 秒，那么每个线程会以 60/10=6（秒/个）的速度连续创建。在 60 秒以后，所有的线程将会投入运行。坡道期设置得足够长可以避免在测试一开始就造成很大的工作负载，可以一开始将坡道设置为要创建的线程数，稍后根据情况进行调整。

Loop Count：定义测试执行的次数，默认情况下，测试仅执行一次。点击 Forever 选项框，测试会一直重复执行直到手动停止它。

Scheduler Configuration：允许设定测试的开始和结束时间。只有时间到达开始时间，测试才会真正开始。在每个测试循环之后，除非结束时间到了，测试将会继续下去一直到达 Loop Count 的上限。Startup delay 用于设定 JMeter 在第一个测试线程启动前的开始等待时间，duration 用于设定整个测试的进行时间。前者用于覆盖 start time 的设定，后者覆盖 end time 的设定。

（2）控制器。JMeter 有两类控制器：Samplers 允许发送特定类型的请求给服务器，稍后我们会发送 http 请求给服务器，所以选用 Http Request 采集器。可以通过 Configuration 元素给采集器进行设定。

Logic Controllers：允许自定义何时发出请求的逻辑。例如可以用 Random Controllers 来向服务器随机发出 http 请求，如下图所示。



（3）采集器。JMeter 采集器允许定义发往服务器的请求，模拟一个用户通过网页向服务器发出的请求。每个采集器生成结果包含性能、延迟时间、吞吐量等。默认情况下，JMeter 会按照采样器出现在测试计划树中的顺序发送请求命令，如下图所示。

3. JMeter 聚合报告

假设在 Web 应用的性能测试中只有一个登录请求，那么在 Aggregate Report 中，会显示一行数据，共有 10 个字段，含义分别如下。

Label：每个 JMeter 的 element(例如 HTTP Request)都有一个 Name 属性，这里显示的就是 Name 属性的值。

#Samples：表示你这次测试中一共发出了多少个请求，如果模拟 10 个用户，每个用户迭代 10 次，那么这里显示 100。

Average：平均响应时间--默认情况下是单个 Request 的平均响应时间，当使用了 Transaction Controller 时，也可以以 Transaction 为单位显示平均响应时间。

Median：中位数，也就是 50%用户的响应时间。

90% Line：90%用户的响应时间。

Min：最小响应时间。

Max：最大响应时间。

Error%：本次测试中出现错误的请求的数量/请求的总数。

Throughput：吞吐量--默认情况下表示每秒完成的请求数（Request per Second），当使用了 Transaction Controller 时，也可以表示类似 LoadRunner 的 Trasaction per Second 数。

KB/Sec：每秒从服务器端接收到的数据量，请求从客户端发出给服务器端之后，服务器会返回给客户端经过处理的结果，这个数据量返回/时间，就是每秒从服务器端接收到的数据量。

实验七 Git 实战

一、实验目的

1）熟练掌握 git 的基本指令和分支管理指令；

2）掌握 git 支持软件配置管理的核心机理；

二、实验内容与步骤

1. 安装 Git

（1）如果在 linux 系统下安装 git，给出安装命令和安装后的运行界面；

（2）如果在 windows 下安装 git，给出安装的 git 版本号和在本地机器上安装 git 后的运行界面（给出主要界面即可）；

（3）对各个界面做出必要的解释。

下载地址： http://git-scm.com/downloads

2. Git 配置

使用 Git 的第一件事就是设置用户的名字和 email，这些就是用户在提交 commit 时的签名。

```
$ git config --global user.name "Scott Chacon"
$ git config --global user.email "schacon@gmail.com"
```

执行了上面的命令后，会在主目录(home directory)建立一个叫 ~/.gitconfig 的文件。内容一般像下面这样:

```
[user]
        name = Scott Chacon
        email = schacon@gmail.com
```

【注:】这样的设置是全局设置，会影响此用户建立的每个项目。

如果想使项目里的某个值与前面的全局设置有区别(例如把私人邮箱地址改为工作邮箱)，那么可以在项目中使用 git config 命令不带 --global 选项来设置。这会在相应的项目目录下的 .git/config 文件增加一节[user]内容(如上所示)。

3. 获得一个 Git 仓库

一切都设置好后，就需要一个 Git 仓库。有两种方法可以得到它：

（1）Clone 一个仓库

为了得一个项目的拷贝(copy)，用户需要知道这个项目仓库的地址(Git URL)。Git 能在许多协议下使用，所以 Git URL 可能以 ssh://，http(s)://，git://，或只是以一个用户名（git 会认为这是一个 ssh 地址）为前辍。有些仓库可以通过不只一种协议来访问，例如，Git 本身的源代码你既可以用 git:// 协议来访问：

```
git clone git://git.kernel.org/pub/scm/git/git.git
```

也可以通过 http 协议来访问：

```
git clone http://www.kernel.org/pub/scm/git/git.git
```

git://协议较为快速和有效，但是有时必须使用 http 协议，比如公司的防火墙阻止了非 http 访问请求。执行了上面两行命令中的任意一个后，就会看到一个新目录: 'git',它包含所有的 Git 源代码和历史记录。

在默认情况下，Git 会把"Git URL"里目录名的'.git'的后辍去掉，做为新克隆(clone)项目的目录名: (例如. git clone http://git.kernel.org/linux/kernel/git/torvalds/linux-2.6.git 会建立一个目录叫'linux-2.6')

（2）初始化一个新的仓库

现在假设有一个叫”project.tar.gz”的压缩文件里包含了用户的一些文件，那么可以用下面的命令让它置于 Git 的版本控制管理之下.

```
$ tar xzf project.tar.gz
$ cd project
$ git init
```

Git 会输出:

```
Initialized empty Git repository in .git/
```

仔细观查会发现 project 目录下会有一个名叫”.git” 的目录被创建，这意味着一个仓库被初始化了。

4. Git 的基本工作流程

（1）创建或修改文件

（2）使用 git add 命令添加新创建或修改的文件到本地的缓存区（Index）

（3）使用 git commit 命令提交到本地代码库

（4）使用 git push 命令将本地代码库同步到远端代码库

5. 分支管理

6. 分布式工作流程

（1）假设 Alice 现在开始了一个新项目，在/home/alice/project 建了一个新的 git 仓库(repository)；另一个叫 Bob 的工作目录也在同一台机器，他要提交代码。

Bob 执行了这样的命令:

```
$ git clone /home/alice/project myrepo
```

这就建了一个新的叫"myrepo"的目录，这个目录里包含了一份 Alice 的仓库的克隆(clone)。这份克隆和原始的项目一模一样，并且拥有原始项目的历史记录。Bob 做了一些修改并且提交(commit)它们:

```
(edit files)
$ git commit -a
(repeat as necessary)
```

当他准备好了，他告诉 Alice 从仓库/home/bob/myrepo 中把他的修改给拉 (pull)下来。她执行了下面几条命令:

```
$ cd /home/alice/project
$ git pull /home/bob/myrepo master
```

这就把 Bob 的主(master)分支合并到了 Alice 的当前分支里了。git pull 命令执行两个操作: 它从远程分支(remote branch)抓取修改的内容，然后把它合并进当前的分支。

其后，Bob 可以更新它的本地仓库--把 Alice 做的修改拉过来(pull):

```
$ git pull
```

如果 Bob 从 Alice 的仓库克隆(clone)，那么他就不需要指定 Alice 仓库的地址，因为 Git 把 Alice 仓库的地址存储到 Bob 的仓库配库文件。

当 Bob 打算在另外一台主机上工作，他可以通过 ssh 协议来执行"clone" 和"pull"操作：

```
$ git clone alice.org:/home/alice/project myrepo
```

（2）公共 Git 仓库

开发过程中，通常大家都会使用一个公共的仓库，并 clone 到自己的开发环境中，完成一个阶段的代码后，可以告诉目标仓库的维护者来 pull 自己的代码。

【完整示例】 https://www.shiyanlou.com/courses/4