### 数据结构：作业 我的PlayStation朋友SP2,

James Baumeister March 2017

1. **介绍**

需要一种管理所有PlayStation朋友的方法，您决定构建一个后端系统来添加，删除和维护它们。这个想法是组织你的朋友，所以你可以搜索个人，搜索与你有相同游戏的玩家，确定排名，并查看玩家信息和奖杯。同时，您希望搜索查询快速排除基本结构，如**数组和链表**。决定订购朋友最重要的因素是层次，你建立一个**二叉搜索树**（BST）结构，使用**层次**（实际上是一个层次上的函数，见第4节）作为关键。

在这个任务中，我们将构建一个**BST结构**，每个节点代表一个PlayStation的朋友。每个朋友节点包含有关该**玩家的信息**，包括其**用户名，级别和出生日期，以及附加的游戏数据结构（单链表）和奖杯（ArrayList）**。按照BST的规则，每个朋友都有一个**父节点和两个子节点**，左和右。从任何一个节点，所有节点到左边都是较小的（较低的水平），而所有的节点到右边的都是较大的（较高的水平）。因此，搜索较高级别或较低级别的玩家平均来说是**O（log n）进程。**

此作业由许多部分组成。

**在部分A中**，您将设置基本的类结构，确保测试套件能够无错误地运行。

**在B部分中**，您将实现用户所需的基本结构，以容纳多个Trophy和Game对象。

**在C部分中**，您将创建基于BST的朋友数据库。

**最后，在D部分中**，您将通过实现AVL平衡来提高您的树的效率。

您可以自由地将自己的方法和字段添加到数据库包中的任何类中，但不要更改任何现有的方法原型或字段定义。已经提供了一个测试套件来测试你的类的功能。这些测试将用于标记您的作业，但使用更改的值。这意味着你不能硬编码答案来通过测试。建议您按照以下各节中列出的顺序完成作业。许多后期课程依赖于它们的依赖关系的正确实现。

## 导入到eclipse

作业是作为一个eclipse项目提供的。 您只需要将项目导入现有工作区。 有关视觉指南，请参见图1。 确保已经设置了您的Java JDK，以及junit要运行的两个jar文件。 这可以在项目→属性→Java构建路径→库中找到。 项目中提供了jar文件; 没有必要下载任何其他版本，这样做可能会影响测试环境。

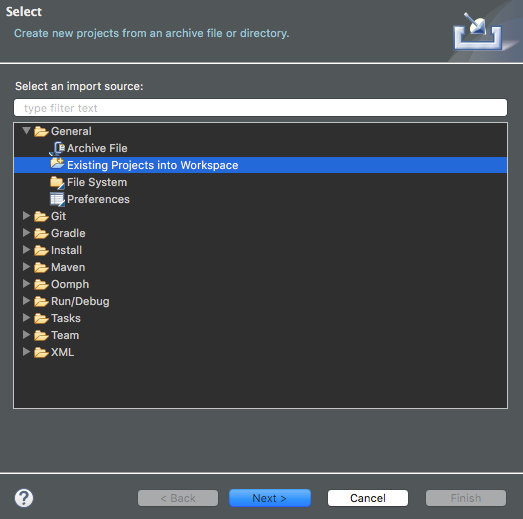


Figure 1: Importing the project through File *→* Import

# Part A

如果您运行测试套件，您将被深深地呈现出许多错误。 您的第一个任务是完成测试期望的类实现（包括实例变量和基本方法），以从测试类中删除所有错误。

## Game

Game类代表一个PlayStation游戏，包括一般信息，即**标题，发布日期和可用奖杯的总数**。 另外，它还引用了**另一个Game对象**。 这将在第3节中重要，您将在其中制作一个单一的游戏列表。 Game类需要以下实例变量:

privateString name; privateCalendar released; privateGame next; privateinttotalTrophies;

toString方法应输出以下格式的字符串（包括引号）：

"Assassin's Creed IV: Black Flag", released on: Nov 29, 2013

提示：打印的日历对象可能看起来不太可能。 看看java日期格式化程序的API。

您还应该生成适当的访问器和mutator方法。

GameTester will assign marks as shown in Table [1:](#_bookmark1)

Table 1: GameTester mark allocation

Test Marks

testConstructor 2

toStringTest 3

**Total:** **5**

## 奖杯Trophy

Trophy类代表一个PlayStation奖杯，并包括有关其**名称，获得的日期和从中获得的游戏的信息**。 此外，其**等级**和**稀有值**是通过枚举器变量提供的有限选项设置的。 奖杯课需要以下实例变量:

publicenumRank{ BRONZE,SILVER,GOLD,PLATINUM

}

publicenumRarity{ COMMON,UNCOMMON,RARE,VERY\_RARE,ULTRA\_RARE

}

privateString name; privateRank rank; privateRarity rarity; privateCalendar obtained; privateGame game;

The toString method should output a string in the following format (quo- tation marks included):

"What Did You Call Me?", rank: BRONZE, rarity: RARE, obtained on: May 04, 2014

您还应该生成适当的访问器和mutator方法。

GameTester will assign marks as shown in Table [2:](#_bookmark2)

Table 2: TrophyTester mark allocation

Test Marks

testConstructor 2

toStringTest 3

**Total:** **5**

## User

User类代表一个PlayStation用户，更一般地说是一个树节点。 最重要的是当使用树节点时，**类必须有一个可以在其上订购树的键**。 在我们的例子中，**它是一个双重命名的键**。 该密钥是基于用户的**用户名**和**级别**的组合的简单功能。 由于级别是整数，并且可能不是唯一的，所以使用简单的方法（参见下面的calculateKey片段）将两个值组合成一个键，同时保持该级别。 例如，假设用户名“abc”的哈希码为1234，用户的级别为3.我们不想将哈希添加到该级别，因为不会保留该**级别**，并导致不正确的排名。 相反，我们计算1234/10000得到0.1234。 然后可以将其添加到该级别。 由于用户名必须是唯一的，所以我们的节点键现在也是独一无二的，用户级别被保留.

privatedoublecalculateKey(){ inthash=Math.abs(username.hashCode());

*// Calculate number of zeros we need*

intlength=(int)(Math.log10(hash)+1);

1A string’s hash value can never be guaranteed to be unique, but for the purposes of this assignment we will assume them to be.

*// Make a divisor 10^length*

doubledivisor=Math.pow(10,length);

*// Return level.hash*

returnlevel+hash/divisor;

}

The User class requires the following instance variables:

privateString username; privateintlevel; privatedoublekey; privateArrayList<Trophy>trophies; privateGameList games; privateCalendar dob;

privateUser left; privateUser right;

选择一个ArrayList类型变量trophies，你想通它会更容易到一个新的奖杯添加到列表中比标准阵列，你可能会大多只是遍历顺序列表中。 选择GameList对象（见第3节）作为存储游戏的结构，因为自定义单链表更适合于编写可重用的方法。

The toString method should output a string in the following format:

User: Pippin

Trophies:

"War Never Changes", rank: BRONZE, rarity: COMMON, obtained on: Mar 26, 2017

"The Nuclear Option", rank: SILVER, rarity: UNCOMMON, obtained on: Mar 26, 2017

"Prepared for the Future", rank: GOLD, rarity: UNCOMMON, obtained on: Mar 26, 2017

"Keep", rank: SILVER, rarity: RARE, obtained on: Mar 26, 2017

Games:

"Yooka-Laylee", released on: May 11, 2017

"Mass Effect Andromeda", released on: Apr 21, 2017 "Persona 5", released on: May 04, 2017

Birth Date: May 23, 1980

You should also generate the appropriate accessor and mutator methods.

UserTester will assign marks as shown in Table [3:](#_bookmark5)

# Part B

在本节中，您将完成GameList单个链接列表，用于存储Game

对象。

Table 3: UserTester mark allocation

Test Marks

testConstructor 2

toStringTest 3

**Total:** **5**

## GameList

GameList类提供了一组用于查找已经链接到形成单链表的Game对象的方法，如图2所示。头是对第一个Game节点的引用，每个Game存储对下一个 游戏，如果游戏结束，则为null.

null

Far Cry 4

Just Cause 3

Tropico 5

head

Figure 2: The single-linked list structure built in GameList

The GameList class requires only one instance variable:

publicGame head

您必须完成许多方法才能接收本节的标记。 他们可以按任何顺序完成。 以下各节概述了每种方法的任务。.

* + 1. voidaddGame(Game game)

此方法应将提供的游戏添加到链表的末尾。 它应该搜索第一个可用的插槽，并适当地设置前一个游戏的下一个变量。 所有的游戏必须是独一无二的，所以你应该检查一下，还没有添加相同的游戏。 请注意，测试需要添加所提供的Game对象，而不是副本。 如果GameList头变量为null，则应更新头来引用新游戏。 如果提供的Game对象为null，则应抛出IllegalArgumentException异常。.

* + 1. GamegetGame(String name)

getGame应遍历链接列表以找到具有匹配名称的游戏。 如果游戏无法找到，该方法应该返回null。 如果提供的名称为null，则该方法应该抛出IllegalArgumentException。.

* + 1. voidremoveGame(String name) **—** voidremoveGame(Game game)

有两个重载的removeGame方法，一个取一个String作为参数，另一个是Game。 这两种方法都应该搜索目标游戏的链表，并将其从列表中删除。 您应该适当地设置前一个节点的下一个变量或设置头变量（如果适用）。 如果它们的参数为空，则两个方法都应该抛出一个IllegalArgumentException异常。.

* + 1. StringtoString()

This method should output a string in the following format:

"Assassin's Creed IV: Black Flag", released on: Nov 29, 2013 "Child of Light", released on: May 01, 2014

## 3.2 GameListTester

GameListTester will assign marks as shown in Table [4:](#_bookmark7)

Table 4: GameListTester mark allocation

Test Marks

getGameNullArg 1

getGame 2

addGame 2

addGameExists 1

addGameNullArg 1

removeGameNullArg 1

removeGameString 2

removeGameObject 2

toStringTest 3

**Total:** **15**

# Part C

在本节中，您将完成您的二进制搜索树数据结构，以存储您的所有朋友的信息.

## BinaryTree

现在所有额外的设置都已经完成了，它们都可以汇集在一起形成你的树结构。 BinaryTree类提供了一组用于形成和更改树的方法，以及一组用于查询树的方法。 目标是形成一个遵守BST规则的树，从而产生一个如图3所示的结构。

User

key = 7.19 (Oberon)

User

key = 4.67 (Faust)

User

key = 10.19

(Pippin)

User

key = 3.16 (Cosette)

User

key = 6.10 (Sophronia)

User

key = 8.16 (Medraut)

User

key = 12.20

(Lunete)

User

key = 1.19 (Nelida)

User

key = 3.20 (Dulcinea)

User

key = 5.21 (Haidee)

User

key = 9.28 (Astaroth)

User

key = 14.20 (Guiomar)

Figure 3: The binary search tree structure built in BinaryTree

The BinaryTree class requires only one instance variable:

publicUser root

There are a number of methods that you must complete to receive marks for this section. They can be completed in any order. Your tasks for each method are outlined in the following sections. Remember that you can add any other methods you require, but do not modify existing method signatures.

* + 1. booleanbeFriend(User friend)
    2. beFriend方法作为参数添加新的用户添加到您的数据库。 遵守BST规则，您应该遍历树，找到正确的位置来添加新朋友。 您还必须更正，根据需要设置左，右和父变量。 请注意，测试要求您添加提供的用户对象，而不是副本。 如果用户密钥已经存在于树中，则此方法应返回false。 如果User参数为null，则此方法应该引发IllegalArgumentException。 例如，将图6中带有密钥6的用户添加到图4中。[.](#_bookmark10)
    3. booleandeFriend(User friend)

deFriend方法将用户从数据库中删除。 该方法应该搜索目标朋友的树并重新移动它们。 这应该通过删除对用户的所有引用和更新左，右和父值（如适用）来实现。 如果朋友成功删除，deFriend应该返回true，否则发现错误或其他错误的情况。 如果friend对象为null，则应抛出IllegalArgumentException异常。 作为示例，从图3中用键4移除用户将导致图5所示的树。

User

key = 7.19 (Oberon)

User

key = 4.67 (Faust)

User

key = 10.19

(Pippin)

User

key = 3.16 (Cosette)

User

key = 6.10 (Sophronia)

User

key = 8.16 (Medraut)

User

key = 12.20

(Lunete)

User

key = 1.19 (Nelida)

User

key = 3.20 (Dulcinea)

User

key = 5.21 (Haidee)

User

key = 6.97 (Bob)

User

key = 9.28 (Astaroth)

User

key = 14.20 (Guiomar)

Figure 4: The BST after adding a friend with key 6

User

key = 7.19 (Oberon)

User

key = 3.20 (Dulcinea)

User

key = 10.19

(Pippin)

User

key = 3.16 (Cosette)

User

key = 6.10 (Sophronia)

User

key = 8.16 (Medraut)

User

key = 12.20

(Lunete)

User

key = 1.19 (Nelida)

User

key = 5.21 (Haidee)

User

key = 9.28 (Astaroth)

User

key = 14.20 (Guiomar)

Figure 5: The BST after removing the friend with key 4

* + 1. intcountBetterPlayers(User reference)

countBetterPlayers方法作为参数，您可以从中搜索具有较高排名的玩家。 这种方法应该从参考用户中搜索并增加更好的玩家的计数器返回。 你应该返回更好的球员数，0如果没有。 请注意，更大的键值不一定等于更高的级别。 如果User参数为null，则此方法应该引发IllegalArgumentException。 例如，使用User与图3中的键7，此方法应返回5.

* + 1. intcountWorsePlayers(User reference)

countWorsePlayers方法将用户作为参数，从中搜索较低等级的玩家。 这种方法应该从参考用户中搜索并增加更糟糕玩家的计数器返回。 你应该返回更糟的球员数量，如果没有，你应该返回0。 请注意，较低的键值不一定等于较低的水平。 如果User参数为null，则此方法应该引发IllegalArgumentException。 作为示例，使用图3中的键7的User，此方法应返回4。.

* + 1. UsermostPlatinums()

mostPlatinums方法应该搜索数据库，并返回具有最多铂金奖杯的玩家。 如果有多名玩家拥有相同数量的铂金奖杯，则应使用金牌打破领带。 如果没有铂金奖杯的用户，此方法应返回null。.

* + 1. voidaddGame(String username,Game game)

addGame方法有两个参数，一个是String用户名和一个Game游戏。 您应该在数据库中搜索匹配的用户，并将新游戏添加到其GameList中。 您还应该检查他们在收集中是否还没有该游戏。 如果任一参数为空，则此方法应抛出IllegalArgumentException异常.

* + 1. voidaddTrophy(String username,Trophy trophy)

addTrophy方法有两个参数，一个是String用户名和Trophy奖杯。

您应该在数据库中搜索匹配的用户，并将新奖杯添加到他们的奖杯中。 你还应该检查一下，他们还没有添加该病毒，并且他们还没有为奖杯的游戏提供所有可用的奖杯。 如果任一参数为空，则此方法应抛出IllegalArgumentException异常.

* + 1. voidlevelUp(String username)

levelUp方法将参数用作String用户名，用于在数据库中搜索匹配的用户。 然后，您应该将该用户的级别增加一级。 如果这违反了任何BST规则，您应该对树进行必要的调整。 作为示例，图6显示了升级后的无效树，图7显示正确的更改。 如果username参数为null，则该方法应该抛出IllegalArgumentException.

User

key = 8.19 (Oberon)

User

key = 3.20 (Dulcinea)

User

key = 10.19

(Pippin)

User

key = 3.16 (Cosette)

User

key = 6.10 (Sophronia)

User

key = 8.16 (Medraut)

User

key = 12.20

(Lunete)

User

key = 1.19 (Nelida)

User

key = 5.21 (Haidee)

User

key = 9.28 (Astaroth)

User

key = 14.20 (Guiomar)

Figure 6: Invalid BST after level-up applied. The key generator allows for multiple level-8 users.

## 4.2 BinaryTreeTester

BinaryTreeTester will assign marks as shown in Table [5.](#_bookmark14)

# Part D

在最后一节中，您将实现AVL树平衡算法。 这将为您的树提供更多的效率，因为它会保持一个完美的平衡，因为不同的值被添加.

### booleanaddAVL(User friend)

addAVL方法将您添加到树中的用户朋友作为参数。 AVL规则应该适用，这意味着如果树变得不平衡，应该进行轮换纠正。 这个优秀的可视化可以帮助您了解如何实现可能需要的任何轮换：https://www.cs.usfca.edu/ galles / visualization / AVLtree.html这个问题在测试文件中分为几个阶段。 测试仅用于升序值，这意味着它们只测试左旋。 替代测试也将测试正确的旋转，所以一定要测试添加降序值。 如果friend参数为null，则此方法应该抛出IllegalArgumentException.

* 1. **AVLTester**

Marks for AVLTest will be assigned as shown in [Table 6.](#_bookmark15)

User

key = 6.10 (Sophronia)

User

key = 3.20 (Dulcinea)

User

key = 10.19

(Pippin)

User

key = 3.16 (Cosette)

User

key = 5.21 (Haidee)

User

key = 8.16 (Medraut)

User

key = 12.20

(Lunete)

User

key = 1.19 (Nelida)

User

key = 9.28 (Astaroth)

User

key = 14.20 (Guiomar)

User

key = 8.19 (Oberon)

Figure 7: Correct operations applied after level-up to preserve BST structure.

Table 5: BinaryTreeTester mark allocation

|  |  |
| --- | --- |
| Test | Marks |
| beFriendNullArg | 1 |
| beFriendDuplicate | 1 |
| beFriend | 6 |
| deFriendNullArg | 1 |
| deFriendNonExistent | 1 |
| deFriend | 7 |
| countBetterPlayersNullArg | 1 |
| countBetterPlayersNonExistent | 1 |
| countBetterPlayers | 4 |
| countWorsePlayersNullArg | 1 |
| countWorsePlayersNonExistent | 1 |
| countWorsePlayers | 4 |
| mostPlatinums | 4 |
| addGameNullArg | 1 |
| addGame | 4 |
| addTrophyNullArg | 1 |
| addTrophy | 4 |
| levelUpNullArgs | 1 |
| levelUp | 7 |
| toStringTest | 3 |
| **Total:** | **54** |

Table 6: avlTester mark allocation

|  |  |
| --- | --- |
| Test | Marks |
| avlStage1 | 6 |
| avlStage2 | 5 |
| avlStage3 | 5 |
| **Total:** | **16** |