

THE UNIVERSITY  
of ADELAIDE

4410\_CO... &gt; Assignments

&gt; Practical 2 - SOLID and ADT design

2024 Semester 1

Home

Search

Announcements

Piazza

Assignments

Echo 360

Grades

Modules

People

Course Readings

Syllabus

Assignment Help

SELT

LMS Analytics

## Practical 2 - SOLID and ADT design

Due Monday by 9:00 Points 6 Submitting an external tool

### General Instructions

This practical is collaborative. You are encouraged to discuss and develop your practicals with your peers, tutor and through Piazza.

Although you are encouraged to work with others, you need to ensure that:

1. the work you submit is work that you have contributed to and understand. You must not just submit the solution of someone else. You must be able to explain any work you submit. If you are unable to explain your submitted work, your tutor is expected to remove any credit awarded.
2. You add a comment at the start of your work identifying other students you worked with

Tutors are available during practical sessions if you get stuck or have questions on any part of this practical.

#### SOLID/ADT Object Oriented Monkey-Robot-Pirate-Ninja-Zombie (moropinzee)

##### Objective

The goal of this practical is to demonstrate that you are able to apply SOLID and ADT design principles in an object oriented application..

##### Problem

In the previous practical, you created a Rock Paper Scissors game. But you've now discovered the far more interesting Monkey-Robot-Pirate-Ninja-Zombie (moropinzee) game <https://markarayner.com/how-to-play-monkey-robot-pirate-ninja-zombie/>. This game is just like Rock, Paper, Scissors but with more options. You can find the details of who wins for each combination of choices on the linked website.

##### SOLID design

To extend your referee to handle these new options, a natural tendency might be to just add more if/then statements to your refGame() function; but as discussed in lecture 3 and in this week's workshop, this violates SOLID design principles and therefore is NOT an acceptable solution for this practical.

Applying your knowledge of SOLID, modify your refGame() so that it complies with SOLID principles. Your refGame() should be able to work **without modification** for any types of moves you might decide to add in the future. It should also continue to support the Rock, Paper and Scissor options. Consider how you can use Polymorphism to achieve this.

Your classes must provide the following interfaces (**changes from previous practical are in bold**).

##### Player (abstract class)

**Move \* makeMove();** // return type is changed, the return type of this is a hint to the polymorphism  
string getName(); // returns the name of the Player as a string

**IMPORTANT CHANGE:** since we now have different moves that start with the same letter, when a move is typed on the keyboard, the user will now enter the full name of the move. For example:

Enter Move: Ninja

The list of possible values the user might enter are:

Rock, Paper, Scissors, Robot, Monkey, Pirate, Ninja, Zombie

##### Referee

##### Referee(); // constructor

Player \* refGame(Player \* player1, Player \* player2)

// returns the reference to the winning player

##### Move

**string getName();** // returns the name of a Move instance, for example "Ninja". This function exists so I can do some tests. You are not required to call it in your code but you may find it useful.

Your submission must contain the following, *but can, and should, also contain other class(es)*.

Move.cpp  
Move.h

Human.cpp  
Human.h

Computer.cpp  
Computer.h

Player.cpp  
Player.h

Referee.cpp  
Referee.h

### ADT design

In addition to the above, each of your classes must adhere to ADT principles as discussed in the second week 2 lecture (ADTs are also in further reading).

##### The main

The main program will create the referee and players, and then ask the referee to adjudicate a game. The main function may create any number of players/referees and call play as many times as it wants. Human players may choose any of the valid moves. Behaviour when invalid moves or incompatible moves (robot and rock for example) are entered is undefined (ie we won't be testing for specific behaviour if a user enters invalid or incompatible moves).

Main will then get the name of the winner using getName() on the returned winning Player pointer and then output this name.

In testing, your code will be tested against our own testing main which will only compile if your code provides the specified interface.

##### Marking Scheme

- Functionality (3 marks):
  - Classes implement the interface - ie they will compile with a main that uses the given interface
  - Passing automark tests
- Design (3 marks) - this will be hand checked as it can not be checked automatically
  - code is correctly indented, has meaningful variable names and is generally readable (1 mark)
  - refGame() does NOT violate the Open Closed principle if additional moves are added to the game (1 mark)
  - All classes adhere to information hiding principles of Abstract Data Types (1 mark)

THE UNIVERSITY  
of ADELAIDE

2024 第一学期

家

搜索

公告

广场

作业

回声 360

成绩

模块

人

课程阅读

教学大纲

作业帮助

塞尔特

LMS 分析

## 实用 2 - SOLID 和 ADT 设计

由于 周一 9: 00 要点 6 提交 外部工具

### 一般说明

这种实践是协作的。我们鼓励您与同龄人、导师和通过Piazza讨论和发展您的实践。

尽管我们鼓励您与他人合作，但您需要确保：

1. 您提交的作品是您贡献和理解的作品。您不能只提交别人的解决方案。您必须能够解释您提交的任何作品。如果您无法解释您提交的作品，您的导师将取消任何授予的学分。
2. 您在工作开始时添加一条评论，以标识与您一起工作的其他学生

如果您遇到困难或对实践的任何部分有疑问，可以在实践课程期间提供导师。

#### SOLID/ADT 面向对象的猴子-机器人-海盗-忍者-僵尸 (moropinzee)

##### 目的

本实践的目的是证明您能够在面向对象的应用程序中应用 SOLID 和 ADT 设计原则。

##### 问题

在之前的实践中，您创建了一个石头剪刀布游戏。但您现在已经发现了更有趣的猴子-机器人-海盗-忍者-僵尸 (moropinzee) 游戏<https://markarayner.com/how-to-play-monkey-robot-pirate-ninja-zombie/>这个游戏就像石头、纸、剪刀一样，但有更多的选择。您可以在链接的网站上找到每种选择组合的获胜者的详细信息。

##### SOLID设计

为了扩展你的裁判来处理这些新选项，一个自然的趋势可能是向你的 refGame() 函数添加更多的 if/then 语句；但正如第 3 讲和本周研讨会所讨论的那样，这违反了 SOLID 设计原则，因此对于这种实用性来说不是一个可接受的解决方案。

运用你对 SOLID 的了解，修改你的 refGame() 使其符合 SOLID 原则。你的 refGame() 应该能够**不加修改地**工作，用于您将来可能决定添加的任何类型的动作。它还应该继续支持石头、纸和剪刀选项。考虑如何使用多态性来实现此目的。

您的类必须提供以下接口（与以前的实际操作相比，更改以粗体显示）。

Move.cpp  
Move.h

Human.cpp  
Human.h

Computer.cpp  
Computer.h

Player.cpp  
Player.h

Referee.cpp  
Referee.h

### ADT设计

除上述内容外，您的每个班级都必须遵守第二周第 2 讲中讨论的 ADT 原则（ADT 也在进一步阅读中）。

##### 主要

主程序将创建裁判和球员，然后要求裁判裁决一场比赛。main 函数可以创建任意数量的球员/裁判，并根据需要多次调用比赛。人类玩家可以选择任何有效的动作。输入无效动作或不兼容动作（例如机器人和岩石）时的行为是未定义的（即，如果用户输入无效或不兼容的动作，我们将不会测试特定行为）。

然后，Main 将在返回的获胜玩家指针上使用 getName() 获取获胜者的名称，然后输出此名称。

在测试中，您的代码将针对我们自己的测试主线进行测试，该主线仅在您的代码提供指定接口时才会编译。

##### 评分方案

- 功能 (3 分):
  - 类实现接口 - 即它们将使用使用给定接口的 main 进行编译
  - 通过自动标记测试
- 设计 (3 分) - 这将被手动检查，因为它不能自动检查
  - 代码缩进正确，具有有意义的变量名称，并且通常可读 (1 标记)
  - refGame() 如果向游戏中添加额外的动作，则不违反 Open Closed 原则 (1 分)
  - 所有类都遵循抽象数据类型的信息隐藏原则 (1 分)

THE UNIVERSITY  
of ADELAIDE

2024 第一学期

家

搜索

公告

广场

作业

回声 360

成绩

模块

人

课程阅读

教学大纲

作业帮助

塞尔特

LMS 分析

## 实用 2 - SOLID 和 ADT 设计

由于 周一 9: 00 要点 6 提交 外部工具

### 一般说明

这种实践是协作的。我们鼓励您与同龄人、导师和通过Piazza讨论和发展您的实践。

尽管我们鼓励您与他人合作，但您需要确保：

1. 您提交的作品是您贡献和理解的作品。您不能只提交别人的解决方案。您必须能够解释您提交的任何作品。如果您无法解释您提交的作品，您的导师将取消任何授予的学分。
2. 您在工作开始时添加一条评论，以标识与您一起工作的其他学生

如果您遇到困难或对实践的任何部分有疑问，可以在实践课程期间提供导师。

#### SOLID/ADT 面向对象的猴子-机器人-海盗-忍者-僵尸 (moropinzee)

##### 目的

本实践的目的是证明您能够在面向对象的应用程序中应用 SOLID 和 ADT 设计原则。

##### 问题

在之前的实践中，您创建了一个石头剪刀布游戏。但您现在已经发现了更有趣的猴子-机器人-海盗-忍者-僵尸 (moropinzee) 游戏<https://markarayner.com/how-to-play-monkey-robot-pirate-ninja-zombie/>这个游戏就像石头、纸、剪刀一样，但有更多的选择。您可以在链接的网站上找到每种选择组合的获胜者的详细信息。

##### SOLID设计

为了扩展你的裁判来处理这些新选项，一个自然的趋势可能是向你的 refGame () 函数添加更多的 if/then 语句；但正如第 3 讲和本周研讨会所讨论的那样，这违反了 SOLID 设计原则，因此对于这种实用性来说不是一个可接受的解决方案。

运用你对 SOLID 的了解，修改你的 refGame () 使其符合 SOLID 原则。你的 refGame () 应该能够**不加修改地**工作，用于您将来可能决定添加的任何类型的动作。它还应该继续支持石头、纸和剪刀选项。考虑如何使用多态性来实现此目的。

您的类必须提供以下接口（与以前的实际操作相比，更改以粗体显示）。

Player (抽象类)

**Move \* makeMove();** // 返回类型变了，这个的返回类型是对多态性

字符串的提示 getName () ;以字符串形式返回播放器的名称

**重要更改：由于我们现在有不同的动作以相同的字母开头，因此当在键盘上键入动作时，用户现在将输入动作的全名。例如：**

Enter Move: Ninja

**用户可能输入的可能值列表包括：**

石头, 纸, 剪刀, 机器人, 猴子, 海盗, 忍者, 僵尸

##### 裁判

裁判 () ;构造 函数

Player \* refGame (Player \* player1, Player \* player2)

// 返回对获胜玩家的引用