作业10:战舰

这个家庭作业比以前的作业更详细,所以尽早开始。它涉及以下主题:

- 继承和覆盖
- 访问修饰符
- 抽象类（我们将在下一讲中了解这些）
- 二维数组

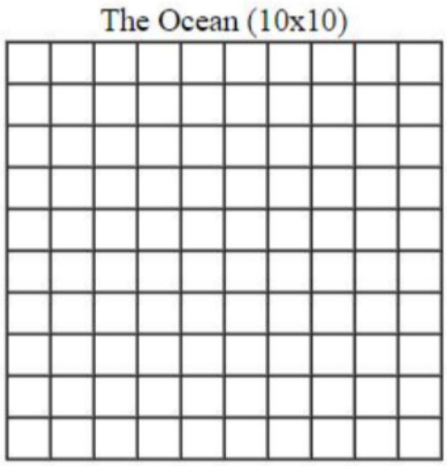
介绍

我们将向您展示如何构建经典游戏Battleship的简单（仅因为没有图形用户界面- GUI)版本。

战舰通常是一个两人游戏,每个玩家都有一支船队和一片海洋（对其他玩家隐藏）,并试图成为第一个击沉其他玩家舰队的人。

我们将只做一个单人与电脑版的游戏,电脑放置船只,人类试图将它们击沉。

我们将在 10x10 的“海洋”上玩这个游戏,并将使用以下船只（“舰队”）：



The Fleet

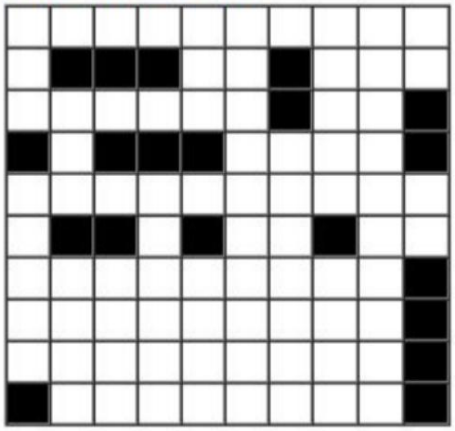
One battleship	■■■■
Two cruisers	■■■ ■■■
Three destroyers	■■ ■■ ■■
Four submarines	■ ■ ■ ■

如何玩战舰

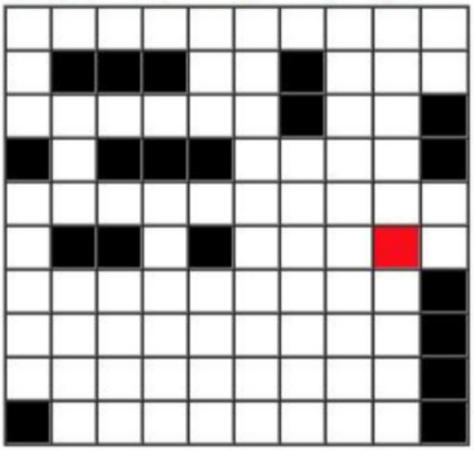
即使您以前玩过《战舰》,也请看看这些规则。
请记住,这是人类与计算机的版本。

计算机将10 艘船放置在海洋上,确保没有船只彼此直接相邻,无论是水平、垂直还是对角线。请查看以下图表,了解合法和非法展示位置的示例:

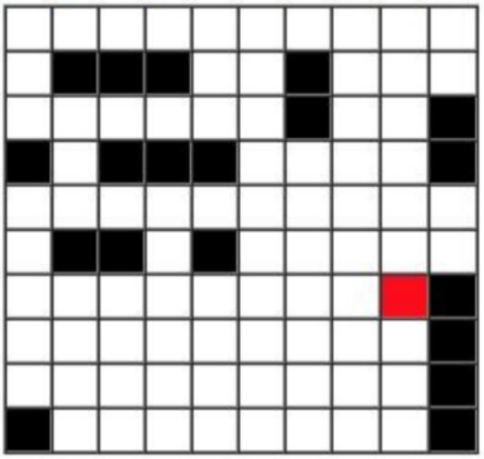
Legal arrangement



Illegal--ships diagonally adjacent



Illegal--ships horizontally adjacent



人类玩家不知道船只在哪里。因此,打印到控制台的海洋初始显示显示一个 10 x 10 的“.”数组。 （有关后续海洋显示的更多信息，请参阅下面对 Ocean 类的 print() 方法的描述）。

人类玩家试图通过指示特定的行号和列号 (r,c) 来击中船只。
计算机以“命中”或“未命中”的信息作为响应。

当一艘船被击中但没有沉没时,该程序不会提供任何关于什么样的船被击中的信息。然而,当一艘船被击沉时,程序会打印出一条消息“你刚刚击沉了一艘船 （类型）”。每次拍摄后,计算机都会用新信息重新显示海洋。

当船的每个方格都被击中时,船就“沉没”了。因此,击沉一艘战舰需要四次（在四个不同的地方）,击沉一艘巡洋舰需要三次,击沉一艘驱逐舰需要两次,击沉一艘潜艇需要一次。

目标是以尽可能少的射击击沉舰队;最好的分数是 20（分数越低越好。）当所有的船都被击沉时,程序会打印出一条消息,游戏结束，并告诉需要多少次射击。

实施细节

命名您的项目Battleship 和您的包战舰。

你的程序应该有以下 8 个类：

- 类战舰游戏
    - o 这是“main”类,包含 main 方法,它首先创建一个海洋实例
- 海洋类
    - o 这包含一个 10x10 的 Ships 数组,代表一个“海洋”,还有一些操纵它的方法
- 抽象类Ship
    - o 这个抽象类描述了所有船舶共有的特征
    - o 它有子类：

- 级战列舰扩展船舶
    - 描述一艘长度为 4 的船
- Cruiser 类扩展 Ship
    - 描述长度为 3 的船
- 类 Destroyer 扩展了 Ship
    - 描述一艘长度为2的船
- Submarine 类扩展 Ship
    - 描述长度为 1 的船
- EmptySea 类扩展 Ship
    - 描述海洋中没有船的部分。 （它
      缺少船是一种船似乎很愚蠢,但这是一种简化很多事情的技巧。这样,海洋中的每个位置都
      包含某种“船”。）

抽象类船

抽象 Ship 类具有以下实例变量。
注意:除非有充分的理由不这样做,否则应将字段声明为私有。这被称为“封装”,这是将类中的字段设为私有并通过公共方法（例如 getter 和 setter)提供对字段的访问的过程。

- private int bowRow
    - o 包含船首的行（船的前部）
- private int bowColumn
    - o 包含船头的柱子（船的前部）
- 私有 int 长度
    - o 船的长度
- 私有布尔水平
    - o 一个布尔值,表示船是水平放置还是水平放置

垂直

- 私有 boolean[] 命中
  - o 一组布尔值,指示船的该部分是否已被击中或
    不是

Ship 类的默认构造函数是:

- public Ship(int length)
  - o 此构造函数设置特定船的长度属性并根据该长度初始化命中数组

Ship类中的方法如下:

吸气剂

- public int getLength()
  - o 返回船长
- public int getBowRow()
  - o 返回与弓的位置对应的行
- public int getBowColumn()
  - o 返回弓柱位置
- public boolean[] getHit()
  - o 返回命中数组
- public boolean isHorizontal()
  - o返回船是否水平

二传手

- public void setBowRow(int row)
  - o 设置 bowRow 的值
- public void setBowColumn(int column)

- 设置 bowColumn 的值

- public void setHorizontal(boolean horizontal)
  - 设置实例变量水平的值

抽象方法

- 公共抽象字符串 getShipType()
  - 以字符串形式返回船舶类型。每种特定类型的船舶（例如 BattleShip、Cruiser 等）必须重写并实现此方法并返回相应的船型。

其他方法

- boolean okToPlaceShipAt(int row, int column, boolean Horizontal, 海洋海洋）
  - 基于给定的行、列和方向，如果可以放置则返回 true 这个长度的船，船头在这个位置；否则为假。该船不得与另一艘船重叠，或接触另一艘船（垂直、水平或对角线），并且不得“伸出”到阵列之外。实际上并没有改变船或海洋 - 它只是说明这样做是否合法。

- void placeShipAt(int row, int column, boolean Horizontal, Ocean 海洋）
  - 将船“放入”大海。这涉及给 bowRow 赋值，bowColumn 和船舶中的水平实例变量，它还涉及在 Ocean 对象的船舶数组中的 1 个或多个位置（最多 4 个）中的每个位置放置对船舶的引用。 （注意：这将有多达四个相同的引用；您不能指代一艘船的“部分”，而只能指整艘船。）
  - 放置一致性（虽然它不会真正影响你如何玩游戏），让我们同意水平船朝东（船首在右端），垂直船朝南（船首在底端）。
    - 这意味着，如果您将一艘水平战舰放置在海洋中的位置 (9, 8)，则船首位于位置 (9, 8)，而船的其余部分位于位置：(9, 7), (9, 6)), (9, 5)。

- 如果您将垂直巡洋舰放置在海洋中的位置 (4, 0),则船首位于位置 (4, 0),而船的其余部分位于位置:(3, 0), (2, 0)。

· boolean shootAt(int row, int column)

o如果船的一部分占据了给定的行和列,并且船没有沉没,则将船的该部分标记为“hit”(在 hit 数组中,索引 0 表示船头)并返回 true;否则返回假。

· 布尔 isSunk()

o 如果船的每一部分都被击中则返回真,否则返回假

· @Override
公共字符串 toString()

o 返回在 Ocean 的 print 方法中使用的单字符字符串。如果船已沉没,则此方法应返回“s”,如果尚未沉没,则应返回“x”。这种方法可以用来打印出海洋中被射击的位置;它不应该用于打印尚未拍摄的位置。由于 toString

所有船类型的行为完全相同,它被放置在 Ship 类中。

类ShipTest

这是 Ship 类的 JUnit 测试类。我们提供的ShipTest.java文件包含该作业自动评分部分的一些单元测试。完整自动评分部分的 Ship 类将进行更多测试。将 ShipTest.java导入您的 Java 项目并在程序的方法中实现足够的代码以通过所有测试。

生成额外的场景并将测试用例添加到每个测试方法中。你应该有一个总
每个方法至少有 3 个不同的场景和有效的测试用例(包括提供的那些)。
例如,一种情况可能是,您通过调用其placeShipAt方法创建战舰并将其放置在海洋中的特定位置。然后创建一个销毁器,并通过调用它的okToPlaceShipAt方法查看它是否可以放置在特定位置。如果是这样,你放置它。然后你创建一个巡洋舰并通过调用它的okToPlaceShipAt方法查看它是否可以放置在特定位置。如果是这样,你放置它。

测试 Ship 类中的每个非私有方法。 (不幸的是,你不能测试私有的方法,因为它们在类之外是不可访问的。)

还要测试 Ship 的每个子类中的方法。您可以在同一个文件中测试 Ship 方法及其子类。

您必须包含注释来解释您的不同测试场景。

类战舰游戏

BattleshipGame 类是“主要”类 也就是说,它包含一个主要方法。在本课程中,您将设置游戏;接受用户的“镜头”;显示结果;并打印最终成绩。所有输入/输出都在这里完成（尽管其中一些是通过调用 Ocean 类中的 print() 方法完成的。）所有计算都将在 Ocean 类和各种 Ship 中完成

类。

为了帮助用户,行号应显示在数组的左边缘,列号应显示在顶部。数字应该是0 到9,而不是 1 到 10。左上角的方块应该是 0、0。使用不同的字符来指示包含命中的位置、包含未命中的位置和从未被射击的位置。

例如,这是程序首次启动时的海洋显示,没有镜头
被解雇了。

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
Enter row,column:
```

使用各种明智的方法。不要把所有的东西都塞进一两种方法中,而是试着把工作分成合理的部分,用合理的名字。

扩展抽象类Ship

使用抽象 Ship 类作为每个船类型的父类。创建以下类并将每个类保存在单独的文件中。

- 级战列舰扩展舰

- 巡洋舰级扩展船舶
- 类驱逐舰扩展船舶
- 潜艇级扩展船舶

这些类中的每一个都有一个零参数公共构造函数,其目的是将长度变量设置为正确的值。从每个构造函数中,使用每艘船的适当硬编码长度值调用超类中的构造函数。注意:您可以将硬编码的 int 值存储在静态最终变量中。

除了构造函数之外,您还必须重写此方法:

- @Override
  公共字符串 getShipType()
    - o 返回字符串“battleship”、“cruiser”、“destroyer”或
      “潜艇”,视情况而定。同样,这些类型的硬编码字符串值非常适合静态最终变量。
    - o 此方法可用于在任何给定时间点识别您正在处理的船舶类型,并且无需使用 instanceof

EmptySea类扩展了Ship

您可能想知道为什么“EmptySea”是一种船。答案是 Ocean 包含一个 Ship 数组,它的每个位置都是(或可以是)对某个 Ship 的引用。如果某个特定位置为空,那么显而易见的做法是在该位置放置一个空值。但是这种明显的方法有一个问题,每次我们查看数组中的某个位置时,我们都必须检查它是否为空。通过在空位置放置一个非空值,表示没有船,我们可以省去所有的空值检查。

EmptySea 类的构造函数是:

- public EmptySea()
    - o 这个零参数构造函数通过调用超类中的构造函数将长度变量设置为 1

EmptySea 类中的方法如下:

- @Override
  布尔 shootAt（int 行,int 列）
    - o 此方法覆盖从 Ship 继承的 shootAt(int row, int column),并且始终返回 false 表示没有命中任何内容

- @Override 布尔
  isSunk()
    - o此方法覆盖从 Ship 继承的 isSunk(),并且始终
      返回 false 表示您没有下沉任何东西

- @Override
  公共字符串 toString()
    - o 返回单字符“-”字符串以在 Ocean 的 print 方法中使用。
      （注意,这是在射击但没有击中时显示的字符。）

- @Override
  公共字符串 getShipType()
    - o 此方法仅返回字符串“empty”

海洋级

实例变量

- private Ship[][]ships = new Ship[10][10]
    - o 用于快速确定哪艘船在任何给定位置
- 私人情报射击开火
    - o 用户射击的总数
- private int hitCount
    - o 子弹击中船只的次数。如果用户多次射击船的同一部分,则每次命中都会被计算在内,即使额外的“命中”对用户没有任何好处。
- 私人情报船沉没
    - o 沉船数量（共 10 艘）

构造函数

- public Ocean()
    - o 创建一个“空”海洋（并用 EmptySea 对象填充船舶数组）。
      您可以创建一个私有辅助方法来执行此操作。
    - o 还初始化任何游戏变量,例如已经发射了多少枪。

方法

- void placeAllShipsRandomly()
    - o 将所有十艘船随机放置在（最初是空的)海洋上。放置更大的船只
      在较小的船之前,否则您可能最终没有合法的地方放置一艘大船。
      您将希望使用 java.util 包中的 Random 类,因此请在Java API 中查找。
    - o 为帮助您编写此方法的代码,请参考 printWithShips()
      下面的方法。它将允许您在编写和调试程序时查看船舶实际放置在海洋中的位置。
- boolean isOccupied(int row, int column)
    - o 如果给定位置包含一艘船,则返回 true,否则返回 false
- boolean shootAt(int row, int column)
    - o 如果给定位置包含一艘“真实”的船,仍然漂浮,则返回 true（不是
      EmptySea）,否则为假。此外,此方法还会更新已开枪的次数和命中次数。
    - o 注意:如果一个位置包含一艘“真实”的船,那么每次用户在同一位置拍摄时,shootAt 都应该返回 true。一旦一艘船“沉没”,在其位置的额外射击应该返回错误。
- int getShotsFired()
    - o 返回射击次数（在游戏中）
- int getHitCount()
    - o 返回记录的命中数（在游戏中）。所有命中都被计算在内,而不仅仅是第一次命中给定的方块。

- int getShipsSunk()
    - o 返回沉没的船只数量（在游戏中）
- boolean isGameOver()
    - o 如果所有船只都沉没则返回真,否则返回假
- Ship[][] getShipArray()
    - o 返回 10x10 的 Ships 数组。 Ship 类中采用 Ocean 参数的方法需要能够查看该数组的内容;这

        placeShipAt() 方法甚至需要修改它。虽然不允许一个类中的方法直接访问另一个类中的实例变量是不可取的,但有时没有好的选择。

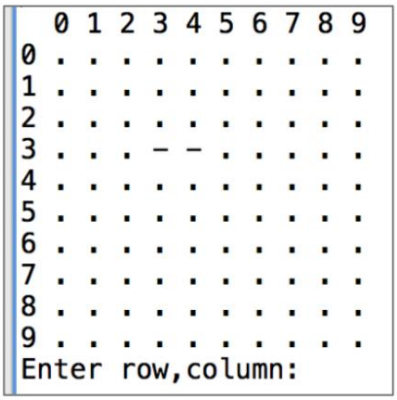
- 无效打印()
    - o 打印海洋。为了帮助用户,行号应显示在数组的左边缘,列号应显示在顶部。

        数字应该是 0 到 9,而不是 1 到 10。

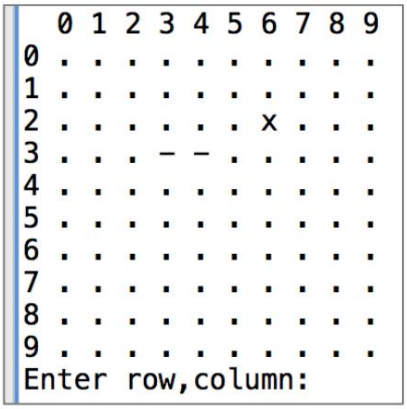
    - o 左上角正方形应该是 0, 0。
    - o x :使用 x 表示您已开火并击中（真实)船只的位置。
      （参考Ship 类中toString的描述）
    - o “-”:使用“-”表示您已开火但没有发现任何东西的位置。 （参考EmptySea 类中toString的描述）
    - o s :使用 s 表示包含沉船的位置。 （参考
      Ship 类中toString的描述）
    - o . :并使用 . （句号)表示您从未开火的地点
    - o 这是 Ocean 类中唯一执行任何输入/输出的方法,它永远不会从 Ocean 类中调用,只能从 BattleshipGame 中调用

        班级。

例如,这里显示了 2 次射门未命中后的海洋。



这是在 2 次射击未命中后的海洋显示,其中 1 次击中了（真正的)船。



为了进一步帮助您了解如何编写此方法的代码,这里是打印海洋的（高级)逻辑：

对于 10 x 10 阵列中的每个位置（“海洋”）

- 如果该位置包含一艘沉没的船,或者该位置是否被击中并被击中
  或者什么也没找到
  - 打印船本身　这将在 Ship 类或任何 Ship 中调用 toString
    定义了 toString 的子类（即 EmptySea）
- 否则打印“.”

- void printWithShips()

  o 仅用于调试目的。

  o 与 print() 方法一样,此方法打印带有行号的 Ocean
  显示在数组的左边缘,列号显示在顶部。数字应该是 0 到 9,而不是 1 到 10。左上角的正方形应该是 0、0。

o 与 print() 方法不同,此方法显示船只的位置。
这种方法可以在开发和调试期间使用,以查看船舶实际放置在海洋中的位置。 (助教在运行你的程序和评分时也可以使用这个方法。)它可以在创建海洋并在其中放置船只后从 BattleshipGame 类中调用。

o 在实际玩游戏和提交您的 Java 项目之前,请务必注释掉对此方法的任何调用。

o b :使用 b 表示战列舰。

o “c”:使用“c”表示巡洋舰。

o “d”:使用“d”表示驱逐舰。

o s :使用 s 表示潜艇。

· :使用 (一个空格)表示EmptySea。

例如,这里是创建海洋后的海洋显示,调用 placeAllShipsRandomly,然后调用 printWithShips。

```
  0 1 2 3 4 5 6 7 8 9
0   d d             s
1
2 c       b b b b
3 c
4 c     s
5                 d
6 d               d   s
7 d     c c c
8
9   s
```

欢迎您编写自己的其他方法。其他方法应该具有默认访问权限(可在包中的任何位置访问)并进行测试,如果您认为它们在此类之外有一些用处。如果您认为它们在本课程之外没有任何用途,请将它们标记为私有。

类OceanTest

这是 Ocean 类的 JUnit 测试类。我们提供的OceanTest.java文件包含该作业自动评分部分的一些单元测试。这里将

对完整自动评分部分的海洋类进行更多测试。将 OceanTest.java导入您的 Java 项目并在程序中的方法中实现足够的代码以通过所有测试。

生成额外的场景并将测试用例添加到每个测试方法中。你应该有一个总
每个方法至少有 3 个不同的场景和有效的测试用例（包括提供的那些）。
例如，一个场景可能是，您通过调用其placeShipAt方法来创建潜艇并将其放置在海洋中的特定位置。然后通过调用isOccupied方法检查海洋中的特定位置是否被占用。通过调用其placeShipAt方法在特定位置创建并放置另一艘潜艇。然后通过调用其isOccupied方法检查海洋中的另一个位置是否被占用。

测试 Ocean 所需的每个方法，包括构造函数，但不包括 print() 方法。如果您在 Ocean 类中创建其他方法，则必须将它们设为私有，或者为它们编写测试。

注意：已经为您完全实现了两种测试方法，这意味着不需要额外的测试用例。 （当然，欢迎您添加更多内容！）。这些是
testEmptyOcean
和testPlaceAllShipsRandomly。

testEmptyOcean测试空海中的每个位置是否都是“空的”。由于 Ocean 类中的构造函数负责创建“空”海洋（并用 EmptySeas 填充 ships 数组），因此可以将testEmptyOcean方法视为

海洋类构造函数。

testPlaceAllShipsRandomly在调用 placeAllShipsRandomly 后测试每个船型的正确数量被放置在海洋中。

您必须包含注释来解释您的不同测试场景。

提交什么

请提交您的 Java 项目中的所有 Java 类。确保将所有内容都包含在“src”文件夹中。

如果您作为团队的一员工作，则只有团队中的一名学生需要提交文件。
将您的团队成员作为@author 标记的一部分包含在每个类的 Javadocs 中。如果 Brandon 与 Sarah 一起工作，他们的 Javadocs 和 @author 标签将类似于：

```
5 /**
6  * Main class for a human vs. computer version of Battleship.
7  * Creates a single instance of Ocean. Gets user input (row and column)
8  * for interacting with and playing against the computer.
9  * @author Brandon Krakowsky & Sarah Broomall
10 */
11 public class BattleshipGame {
```

评估

您将被评为 52 分：

·风格（共 8 分）

o 这包括但不限于：

- 将 Javadocs 添加到每个类、方法和变量，并将注释添加到所有重要的代码。
- 正确缩进（Cmd+i 或 Ctrl+i)并正确使用 {括号} 循环和条件
- 使用 camelCase 描述性地命名其他变量和方法
- 删除未使用的变量并注释掉用于调试的代码块/打印语句

·游戏玩法（共 10 分）

o 这归结为 TA 是否可以玩你的游戏。界面应该清楚。如果您做出了一些可能不寻常的设计选择，请确保你非常清楚地指出这一点。如果你不知道这是什么意味着，可能值得在 EdDiscussion 或办公时间询问。如果您按照信中的所有说明进行操作，那就没问题了。

·单元测试（共 24 分）

o 请确保您通过了提供的测试以及您自己的附加单元测试（10 分）

o 通过我们自己的单元测试（**AUTOGRADED**） （14 分）

o 注意，有些方法不能进行单元测试，例如接受用户输入。同样，打印到控制台（并且仅执行此操作)的方法无法进行单元测试。

o 注意：请清楚地注释您的单元测试以解释不同的场景

你正在测试。助教会阅读你的测试以确保它们是不同的
并且有效。

·代码编写（10分）

- o 确保您了解继承并正确使用覆盖
- o 注意：请清楚地注释您的 placeAllShipsRandomly() 方法。助教将阅读您的这部分代码，并确保您确实正确地执行了它。