Homework 10 : Call of Duty

For HW10, you may work as a group (no more than 2 students). **Please mention your collaborator's name at the top of your code files.**

This homework is more detailed than previous assignments, so start as early as you can on it. It deals with the following topics:

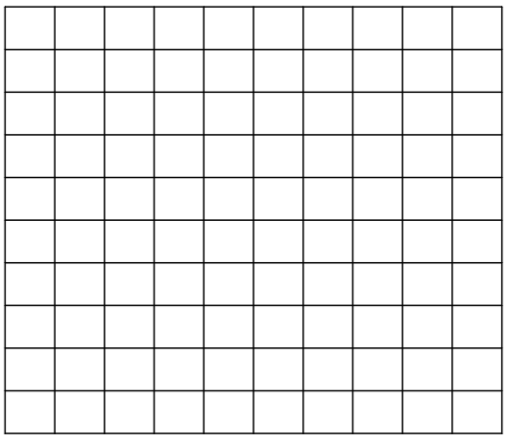
- Inheritance & overriding
- Access modifiers
- Abstract classes (we'll learn about these in the next lecture)
- 2-dimensional arrays

Introduction

We are going to show you how to build a *simple* (only because there is no *graphical user interface* - GUI) version of the classic game **Call of Duty**.

In the game, you serve as a soldier and your mission is destroying a 10x10 enemy base. You need to destroy all Targets in the base.

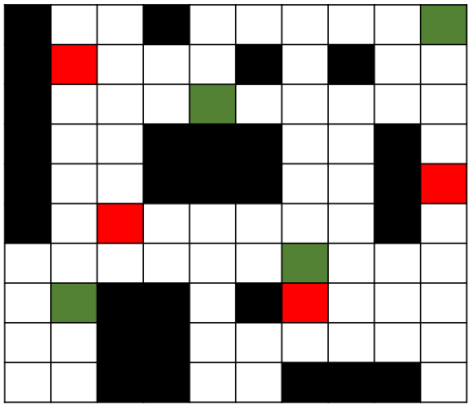
The base 10x10



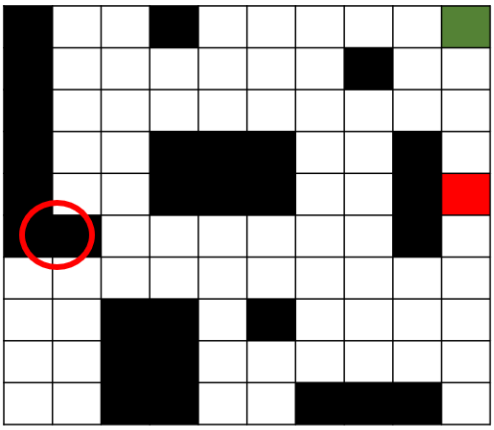
How to Play Call of Duty

Before the game begins, the program places all targets randomly in such a way that no buildings are immediately adjacent to each other, either horizontally, vertically, or diagonally. Take a look at the following diagrams for examples of legal and illegal placements. (Notes: Tanks and oil drums are not buildings, so they could be placed directly adjacent to each other, or to buildings.)

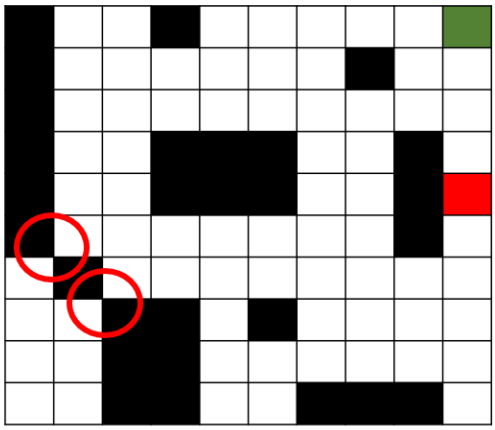
Legal arrangement



Illegal buildings horizontal adjacent



Illegal buildings diagonally adjacent



The player doesn't know where the targets are. The base is initially covered by mist, and the initial display of the base to be printed to the console therefore shows a 10x10 array with a '.' (a period) in every location. (See the description of the Base class' print() method below for more information on what subsequent base displays will look like.)

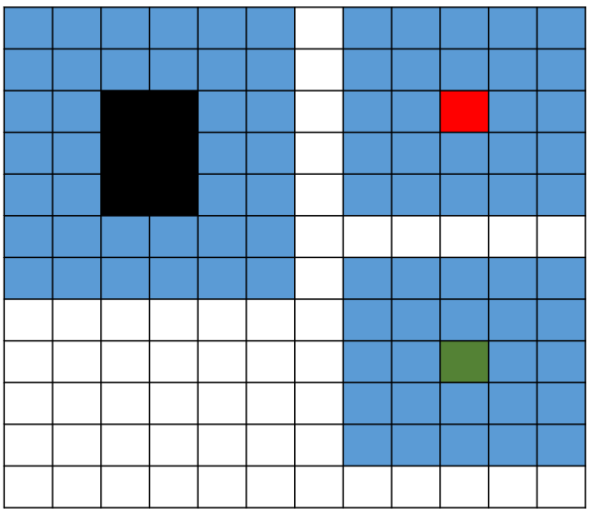
The player tries to hit the targets, by indicating a specific row and column number (r,c). The player has two weapons, a "rocket launcher" and a "missile". An input of "q" will switch the weapon. You can shoot a "rocket launcher" 20 times and shoot a "missile" 3 times. The "rocket launcher" will hit one square at the specific coordinate. However, the "missile" will hit a 3x3 area with the coordinate as its center. The program should display the currently selected weapon.

When a target is hit but not destroyed, the program does not provide any information about what kind of a target was hit. However, when a target is hit and destroyed, the program prints out a message "You have destroyed a xxx." After each shot, the computer refreshes the base, meaning, it prints the entire base (10x10 array) again.

A target is "destroyed" when every square of the target has been hit. But tanks can withstand two hits! Thus, it takes 6 hits to destroy a headquarter and an armory, 3 for barracks, 2 for tanks and 1 for oil drums.

When an oil drum or a tank is destroyed, it will explode and hit a 5x5 area around it. When an armory is destroyed, it will explode and hit a 6x7 (or 7x6) area around it. And the explosion can spread, i.e., an explosion can trigger another explosion.

Explosion range



The object is to destroy all targets with as few shots as possible. But you can only shoot a rocket launcher 20 times and a missile 3 times. If all targets are destroyed before you run out of ammunition, you win. Otherwise, you lose. You can switch weapons. Each time before you shoot, the program tells you how many shots are left.

The program prints out a message indicating whether you have won or lost, and how many shots were required, when the game is over.

Details of Implementation

Name your **project "**CallOfDuty", and your **package "**callOfDuty". Your program should have the following 13 classes:

- Class CallOfDutyGame
  This is the "main" class, containing the main method, which starts by creating an instance of Base.

- Class Base
  This contains a 10x10 array of Targets, representing a "base", and some methods to manipulate it. Think of it as the map in this game.

- Abstract class Target

This abstract class describes the characteristics common to all targets.
It has subclasses:

1) class HeadQuarter extends Target
2) class Armory extends Target
3) class Barrack extends Target
4) class SentryTower extends Target
5) class Tank extends Target
6) class OilDrum extends Target
7) class Ground extends Target

Ground describes a part of the base that doesn't have a target on it. (It seems silly to have the lack of a target be a type of target, but this is a trick that simplifies a lot of things. This way, every location in the base contains a "Target" of some kind.)

- Class Weapon

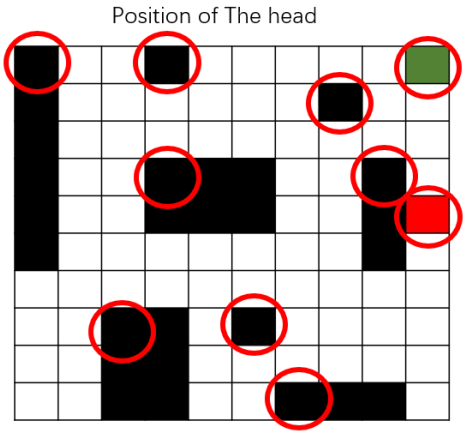
  This abstract class describes the characteristics common to all Weapons.
  It has subclasses:

  1) class RocketLauncher extends Weapon
  2) class Missile extends Weapon

Abstract class Target

The abstract Target class has the instance variables below. Note: Fields should be declared private unless there is a good reason for not doing so. This is known as "encapsulation", which is the process of making the fields in a class private and providing access to the fields via public methods (e.g. getters and setters).

- private int[] coordinate
  - An array of length 2 that specifies the coordinate of the head of a target. Head means the upper left part of a Target.



- private int length

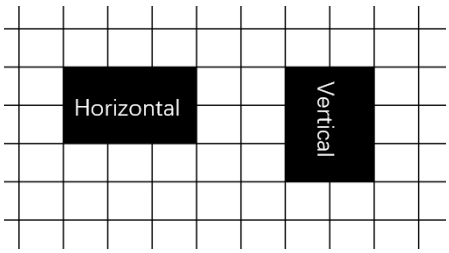
- The length of the Target

- private int width
  - The width of the Target

  width <= length

- private boolean horizontal
  - Indicates whether the Target is horizontal or not

- private int[][] hit
  - An array of the same size as the target, indicating the number of times a part of the target has been hit.
  - For example, an Armory has a length of 3 and a width of 2. A horizontal Armory has a hit of int[2][3] and a vertical Armory has a hit of int[3][2]



- private Base base
  - An instance of Base that the target is placed in.

The default Constructor for the Target class is:

- public Target(int length, int width, Base base)
  - This constructor sets the length, the width and the base of the Target.

The **methods** in the Target class are as follows:

Getters

- public int[] getCoordinate()
  Returns the coordinate array
- public boolean getHorizontal()
  Returns whether the target is horizontal or not
- public int[][] getHit()
  Returns the hit array
- public Base getBase()
  Returns the base
- public int getLength()
  Returns the length of this target
- public int getWidth ()
  Returns the width of this target

Setters

- void setCoordinate(int[] coordinate)
  Sets the coordinate array
- void setHorizontal(boolean horizontal)
  Sets the value of horizontal
- void setHit(int[][] hit)
  Sets the value of hit array

Abstract Methods

- abstract void explode();
  - Defines the behavior when a target is destroyed. Some may explode, while some do nothing.
- public abstract String getTargetName()
  - Returns the type of Target as a String. Every specific type of Target (e.g. Armory, Tank, etc.) has to override and implement this method and return the corresponding Target type. This method is not case sensitive, i.e., “Armory”, “ARMORY” and “armory” are all acceptable.

Other Methods

- public void getShot (int row, int column)
  - If a part of the Target occupies the given row and column and it is not destroyed, mark that part of the Target as “hit” (in the hit array, index (0,0) indicates the head).
- public boolean isDestroyed()
  - Returns true if every part of the Target has been hit, false otherwise.
  - For Tank, every part of it should be hit twice. So override this method in the Tank Class.

- public boolean isHitAt(int row, int column)
  - Returns true if the target has been hit at the given coordinate. This method is used to print the Base.

- public String toString()
  - Returns a single-character String to use in the Base’s print method. This method should return ”X” if the Target has been destroyed and ”O” (capital letter O) if it has not been destroyed. But for an undestroyed Tank, it returns “T”.
  - If the Target is Ground, it returns “-”.
  - This method can be used to print out locations in the Base that have been shot at (including those hit by explosion); it should not be used to print locations that have not been shot at. Since toString behaves exactly the same for all Target types but for Tank and Ground, it is placed here in the Base class. You should override it in Tank and Ground class.

Extending abstract class Target

Use the abstract Target class as a parent class for every single Target type. Create the following classes and keep each class in a separate file.

- class HeadQuarter extends Target
- class Armory extends Target
- class Barrack extends Target
- class SentryTower extends Target
- class Tank extends Target
- class OilDrum extends Target
- class Ground extends Target

Each of these classes has a one-argument (Base base) public constructor, the purpose of which is to set the length, width and base to the correct value. From each constructor, call the constructor in the super class with the appropriate hard-coded width, length and base for each Target. (Note: You can store the hard-coded int values in static final variables.)

Aside from the constructor you have to override the following methods:

- public String getTargetName()
  - Returns one of the strings “headQuarter”, “armory”, “barrack”, “sentryTower”, ”tank”, ”oilDrum”, or “ground”, as appropriate. (Again, these types of hard-coded string values are good candidates for static final variables, and these are case insensitive)
  - This method can be useful for identifying what type of Target you are dealing with, at any given point in time, and eliminates the need to use *instanceof*.

- public void explode()
  - Define how the target will behave when it is destroyed. Only Tank, OilDrum and Armory can explode.

Class Ground **extends** Target

You may wonder why “Ground” is a type of Target. The answer is that the Base contains a Target array, every location of which is (or can be) a reference to some Target. If a particular location is empty, the obvious thing to do is to put a null in that location. But this obvious approach has the problem that, every time we look at some location in the array, we’d have to check if it is null. By putting a non-null value in empty locations, denoting the absence of a Target, we can save all that null checking.

Constructor

- public Ground(Base base)

- This constructor sets the width and length variables to 1 and the Base variable by calling the constructor in the super class.

Special Methods

- @Override
  public boolean isDestroyed()
  This method overrides isDestroyed () that is inherited from Target, and always returns false to indicate that you didn't destroy anything.

- @Override
  public String toString()
  Returns the single-character "-" String to use in the Base's print method. (Note, this is the character to be displayed if a shot has been fired, but nothing has been hit.)

Class TargetTest

We've provided this class for you – import *TargetTest.java* into your Java project. Implement enough code in the methods in your program to pass all tests. Generate additional scenarios and add test cases to each test method. You should have a total of *at least 3 distinct scenarios and valid test cases* per method (including the ones provided).

Test every non-private method in the Target class. (Unfortunately, you can't test methods that are private because they are inaccessible outside of the class.) Also test the methods in each subclass of Target. You can do this here or in separate test classes (e.g. TankTest, ArmoryTest, etc.), if you wish.

No need to test Setters and Getters.

Abstract Class Weapon

The abstract Target class has the instance variables below. Note: Fields should be declared private unless there is a good reason for not doing so. This is known as "encapsulation", which is the process of making the fields in a class private and providing access to the fields via public methods (e.g. getters and setters).

Instance Variables

- private int shotleft
  - The number of shots left. Initially, it's 20 for RocketLauncher and 3 for Missile.

Constructor

- public Weapon(int shotCount);
- public Missile(); for Missile Class
- public RocketLauncher(); for RocketLauncher Class

Getters

- public int getShotLeft()

Abstract Methods

- public abstract String getWeaponType()
  - RocketLauncher will return “rocketLauncher”, and Missile will return “missile” and it is case insensitive.
- public abstract void shootAt(int row, int column, Base base)
  - RocketLauncher will only shoot at one square, while Missile will attack a 3x3 area. It will call shootAt(int row, int column) and incrementshotsCount () method in Base class.

Other Method

- public void decrementShotLeft()

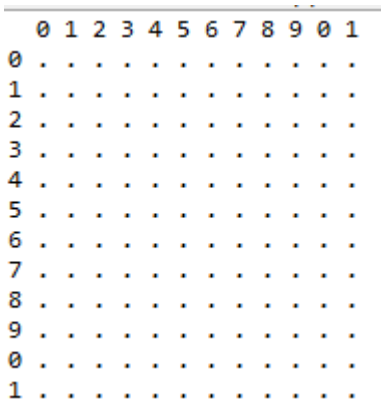
Class WeaponTest

This is the JUnit test class for Weapon. **We’ve provided this class for you** – import *WeaponTest.java* into your Java project. Implement enough code in the methods in your program to pass all tests. Generate a total of *at least 3 distinct scenarios and valid test cases* for each method.

Class CallOfDutyGame

The CallOfDutyGame class is the “main” class -- that is, it contains a main method. In this class you will set up the game; accept ”shots” from the user; display the results; print the final result; and ask the user if he/she wants to play again. All input/output is done here (although some of it is done by calling a print() method in the Base class.) All computation will be done in the Base class and the various Target classes.

To aid the user, row numbers should be displayed along the left edge of the array, and column numbers should be displayed along the top. Numbers should be (**0 to 9)**, not 1 to 10. The top left corner square should be 0, 0. Use different characters to indicate locations that contain a hit, locations that contain a miss, and locations that have never been fired upon. For example, here’s a display of the Base when the program first launches, and no shots have been fired.



Use various sensible methods. Don't cram everything into one or two methods, but try to divide up the work into sensible parts with reasonable names.

Class Base

Instance variables

- private Target[][] targets
  Keeps a reference to the location of every Target in the game. Every location in this array points to a Target, specifically, an instance of a subclass of Target.

- private int shotsCount
  The total number of shots fired by the user.

- private int destroyedTargetCount
  The number of targets destroyed.

Constructor

- public Base()
  Creates an 10x10 "empty" Base (and fills the Targets array with Ground objects). You could create a private helper method to do this. Also initializes any game variables, such as how many shots have been fired.

Methods

- public void placeAllTargetRandomly()
  - **Create and place** all Targets randomly on the Base (initially filled with Ground). Place larger Target before smaller ones, and place buildings before tanks and oil drums, or you may end up with no legal place to put a Target. You will want to use the Random class in the java.util package, so look that up in the Java API.
- public boolean okToPlaceTargetAt(Target target, int row, int column, boolean horizontal)
  - Based on the given row, column, and orientation, returns true if it is

okay to put the Target with its head at this location; false otherwise. The buildings must not overlap another Target, or touch another building (vertically, horizontally, or diagonally). And targets must not "stick out" beyond the base. Does not actually change either the Target or the Base – it just says if it is legal to do so.

- public void placeTargetAt(Target target, int row, int column, boolean horizontal)
  - Sets the value of the "hit" array, "coordinate" array, and "horizontal" boolean value of the target.
  - "Put" the Target in the Base. This involves giving values to the coordinate and horizontal instance variables in the Target, and it also involves putting a reference to the Target in each of 1 or more locations in the targets array in the Base object. (Note: There will be many identical references; you can't refer to a "part" of a Target, only to the whole Target. For example, an Armory occupies a 2x3 area in the base, and all of these 6 squares reference to the same instance of an Armory)
  - For placement consistency (although it doesn't really affect how you play the game), let's agree that the head of a Target is the upper left part of it.
    - This means, if you place a horizontal Armory at location (5, 5) in the Base, the head is at location (5, 5) and the rest of the it occupies locations: (5, 6), (5, 7),(6, 5),(6,6),(6,7)
    - If you place a vertical Armory at location (5, 5) in the Base, the head is at location (5, 5) and the rest of the Target occupies locations: (5, 6),(6, 5),(6, 6),(7,5),(7,6),
- public boolean isOccupied(int row, int column)
  - Returns true if the given location contains a Target(not a Ground), false if it does not.

- public void shootAt(int row, int column)
  - Attack the position specified by the row and the column.
- public boolean isGameOver(Weapon weapon1, Weapon weapon2)
  - Returns true if run out of ammunition or if all targets have been destroyed. Otherwise return false.

- public boolean win()
  - Returns true if all targets have been destroyed.

- public void print()
  - Prints the Base. To aid the user, row numbers should be displayed along the left edge of the array, and column numbers should be displayed along the top. Numbers should be (0 to 9), not 1 to 10.

- The top left corner square should be 0,0.
- "O" (capital letter O): Used to indicate a location that you have fired upon and hit a target (reference the description of toString in the Target class)
- "-": Use '-' to indicate a location that you have fired upon and found nothing there (reference the description of toString in the Ground class)
- "X" (capital letter X): Use 'X' to indicate a location containing a destroyed Target.
- '.': Use '.' (a period) to indicate a location that you have never fired upon
- "T": Used to indicate an undestroyed but hit Tank
- This is the only method in the Base class that does any input/output, and it is never called from within the Base class (except possibly during debugging), only from the CallOfDutyGame class.

For example, here's a display of the base after 2 shots have missed:

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
RPG: 20 Missile: 3
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon 2,4
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . - . . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
RPG: 19 Missile: 3
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon 2,5
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . - - . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
RPG: 18 Missile: 3
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon
```

The following is a display when using a missile to shoot at and hit (3,3). A sentry tower at (2,3) is destroyed and a sentry tower at (4,3) is destroyed, because they are both within the scope of the missile. In this case, all targets

from (2,2) to (4,4) is hit.

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . - - . . . .
3 . . . . . . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
RPG: 18 Missile: 3
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon q
RPG: 18 Missile: 3
Your current weapon is: missile
Enter row,column, or q to switch a weapon 3,3
You have destroyed a sentryTower
You have destroyed a sentryTower
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . - X - - . . . .
3 . . - - - . . . . .
4 . . - X - . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
RPG: 18 Missile: 2
Your current weapon is: missile
Enter row,column, or q to switch a weapon
```

You can't shoot when the current weapon runs out of ammunition.

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . - X - - . . . .
3 . . - - - . . . . .
4 . . - X - - 0 . . .
5 . . - T - - 0 . . .
6 . . 0 0 X - 0 - - .
7 . . 0 0 - - - - - .
8 . . - - - - - - - .
9 . . . . . . . . . .
RPG: 18 Missile: 0
Your current weapon is: missile
Enter row,column, or q to switch a weapon 3,6
No ammunition!
RPG: 18 Missile: 0
Your current weapon is: missile
Enter row,column, or q to switch a weapon q
RPG: 18 Missile: 0
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon 3,6
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . - X - - . . . .
3 . . - - - . 0 . . .
4 . . - X - - 0 . . .
5 . . - T - - 0 . . .
6 . . 0 0 X - 0 - - .
7 . . 0 0 - - - - - .
8 . . - - - - - - - .
9 . . . . . . . . . .
RPG: 17 Missile: 0
Your current weapon is: rocketLaucher
Enter row,column, or q to switch a weapon
```

- public int getShotsCount()
  Returns the number of shots fired

- public Target[][] getTargetsArray()
  Returns the targets array

- public void incrementShotsCount()
  This method will be called from shootAt(int row, int column) from Weapon class.

- public int getDestroyedTargetCount()
  Returns the count of destroyed targets

- public void setDestroyedTargetCount(int i)
  Set the count of destroyed targets

You are welcome to write additional methods of your own. Additional methods should have default access (accessible anywhere in the package) and be tested, if you think they have some usefulness outside of this class. If you don't think they have any use outside of this class, mark them as private.

Class BaseTest

This is the JUnit test class for Base. **We've provided this class for you** – import *BaseTest.java* into your Java project. Implement enough code in the methods in your program to pass all tests.

Generate additional scenarios and add test cases to each test method. You should have a total of *at least 3 distinct scenarios and valid test cases* per method (including the ones provided). For example, a scenario could be, you create and place a HeadQuarter in a particular location in the Base by calling its *placeTargetAt* method. Then you check if a particular location in the Base is occupied by calling its *isOccupied* method. Create and place another Tank in a particular location by calling its *placeTargetAt* method. Then check if another location in the Base is occupied by calling its *isOccupied* method.

Test every required method for Base, including the constructor, but not including the print() method. If you create additional methods in the Base class, you must either make them private, **or** write tests for them. Test methods do not need comments, unless they do something non-obvious.

Note: One test method has been **completely implemented for you**, meaning, no additional test cases are required. (Of course, you're welcome to add more!). This is *testPlaceAllTargetsRandomly*. *testPlaceAllTargetsRandomly* tests that the correct number of each Target type is placed in the Base after calling *placeAllTargetsRandomly()*.

No need to test Setters and Getters.

Javadocs

Create *API (Application Programming Interface)* documentation for your entire program. This can be extremely helpful for other programmers reading/running your code. Eclipse makes it very easy to generate HTML files directly from the Javadocs in your code. See the separate "Generating Javadocs.pdf" file for how to do this.

What to Submit

Please submit all the Java classes in your Java project. Make sure to include everything in your "src" folder, and a separate folder with all of your generated Javadoc HTML files.

Evaluation

You will be graded out of 45 points:

- **Style** (*5 pts total*)
  This includes, but is not limited to:
  - Adding Javadocs to methods and variables, and comments to all non-trivial code
  - Indenting properly (Cmd+i or Ctrl+i) and using { brackets } correctly in loops and conditionals
  - Naming additional variables and methods descriptively with camelCase
- **Javadocs** (*5 pts total*)
  - Be sure to write Javadocs for every method and variable
  - Generate HTML files from the Javadocs in your code using Eclipse
- **Game play** (*10 pts total*)
  - This comes down to whether or not a TA can play your game.
  - The interface should be clear. If you have made some potentially unusual design choice, please make sure that you point that out very clearly. If you do not know what this means, it might be worth asking on piazza or during office hours. If you followed all the instructions to the letter, you are fine.
- **Unit testing** (*15 pts total*)
  - Please make sure you pass the provided tests as well as your own additional unit tests (10 pts)
  - Passing our own unit tests (**AUTOGRADED**) (5 pts)
  - Note, there are some methods which cannot be unit tested, such as a method that takes in user input. Similarly, a method that prints to the console (and does only that) cannot be unit tested.
- **Code writing** (*10 pts*)
  - Make sure you understand inheritance and correctly utilize overriding

- Note: Please clearly comment your placeTargetsRandomly() method. The TAs will read this part of your code and make sure that you are actually doing it correctly.