Question 2: Inheritance

1. Instruction

1. Create an Eclipse project named "**Q2**"

2. Copy all folders in folder "**toStudent/Part2_toStudent**" of the given zip file to your project **src** folder.

3. You are required to implement or complete the following classes (details for each class are provided in Section 3.)

   i. GameManager (package **logic.game**)

   ii. HardStone (package **logic.stone**)

   iii. Dynamite (package **logic.stone**)

   iv. WalkingStone (package **logic.stone**)

   v. Gear (package **logic.stone**)

4. You also need to create a UML Diagram containing all classes in the **logic.stone** package. Save it as **umlQ2.png** at the **root** folder of your project.

5. JUnit for testing is in the package "test".

6. Export jar file (**q2.jar**) that **includes your source code and your .class files** and put it in **root** folder of your project. **YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED**.

You should read the whole document before starting code. You may need to use some methods from the already provided classes.

2. Problem Statement: Mining Simulator

In *Mining Simulator*, we aim to develop a grid-based mining game. The player can move around the map and use a drill to mine different types of stones. Each time the player digs, the drill's battery depletes. However, the drill can be upgraded to speed up the digging process. The objective is to achieve the highest possible score before the battery runs out. There are six types of stones:

- Stone [-]: Destroyed in a single dig.
- HardStone [H]: Has durability, making it harder to destroy. Durability decreases with each dig and it is destroyed when durability reaches zero.
- Dynamite <D>: Destroyed in a single dig, but also destroys all adjacent stones upon destruction.
- Gear {G}: Has durability. When destroyed, it increases the drill's power.
- Crystal {C}: Has durability and awards points with each dig attempt. When destroyed, it grants 10 points.
- WalkingStone <W>: Destroyed in a single dig. This stone can move and will chase the player if possible.

```
SCORE: 0
Dig Power: 1
Battery Left: 20
[-][-][-][-][-][-][-][-][-][-][-][-][-][-][-][-]
[-][-][-][-][-][-][-][H][H][H][-][-][-][-][-][-]
[-][-][-][-][-][-][-][-][-][H][-]{G}[-]{C}[-][-]
[-][-][-]{G}[-][-]         [-][-][-][-]<W>[-]
[-][-][-][-][-][-][-]   (P)   [-]<D>[-][-][-][-]
[-][-]{G}[-][-][-]         [-][-][-]{C}[-][-]
[-][-][-]<D>[-][-][-][-][-][-][-][-][-]<W>[-]
[-][-]{C}[-][-][-]{G}[-][-][-][-][-][-][-][-]
[-][-][-][-][-][-][-][-][-][-][-][-][-][-][-]
```

3. Implementation Details

To complete this question, you need to understand about **Inheritance**. The class packages are summarized below. Only the key methods you need to implement or complete for this question are listed. Noted that access modifier notations are listed below

+ (public)

(protected)

- (private)

Underline (static)

Italic (abstract)

3.1 Package application /*Already Provided*/

3.1.1 Class Main /*Already Provided*/

This class is a main program that controls how the game flows. You can run Main.java to see how the program works.

3.2 Package logic /* Partially Provided*/

3.2.1 Package logic.game

3.2.1.1 Class GameManager / Partially Provided*/*

This class manages the game logic

Fields

Field	Description
- GameManager instance	Instance of GameManager

- int digPower	Drill's dig power (higher value will make digging HardStone faster)
- int gameScore	Current game's score
- int batteryLeft	Amount of battery left that player could use to dig.
- boolean isGameEnded	True if the game has ended.

Methods

Method	Description
+ GameManger()	Constructor of this class Set digPower to 1 Set gameScore to 0 Set batteryLeft to 20 Set isGameEnded to false
+ GameManager getInstance()	Create new GameManager if there's no instance otherwise return current instance
+ GameManager resetInstance()	Reset GameManager and return the new one.
+ void doNextStep(char action)	doAction based on input action string Command every walking stone to walk
- void doAction(int x, int y)	If the input coordinates (x, y) correspond to an empty tile, move to that tile. If there is a stone, use digStone() to dig it.
+ void digStone(Stone stone)	// FILL CODE Use the drill's power to dig the stone, which will also reduce the battery by 1. If the stone is Dynamite, 5 scores will be deducted due to the explosive force. Hint#1: Use dig(digPower) of the stone to dig the stone with digPower Hint#2: you can use function useBattery to decrease battery.

+ void endGame()	Set isGameEnded to true
+ void useBattery(int amount)	Decrease batteryLeft by 1 But if batteryLeft is 0, then endGame
+ void addScore(int score)	Increment current score
+ void addDigPower(int amount)	Upgrade current drill's power by some amount
+ getter/setter for some variables	

3.2.2 Package logic.stone /*Partially Provided*/

*3.2.2.1 Class Stone /*Already Provided*/*

This class represents stone.

Fields

Field	Description
# int posX	Current x-axis position of this stone
# int posY	Current y-axis position of this stone

Methods

Method	Description
+ Stone(int posX, int posY)	Constructor of this stone class Set posX and posY
+ void dig(int digPower)	Destroy this stone immediately
+ void destroy()	Remove stone from the map Hints: can remove stone from the map using 'GameUtilities.removeStone(Stone stone)' function
+ getter/setter	

*3.2.2.2 Class HardStone /*you must implement from the scratch*/*

This class represents a stone that's harder to destroy than a normal stone.

Fields

Field	Description
# int durability	How hard this stone is to destroy. This value can't be less than 0 or more than 5

Methods

Method	Description
+ HardStone(int posX, int posY, int durability)	Constructor of this class Set posX, posY and durability
+ void dig(int digPower)	Decrease durability by dig power. But if the durability is less than or equal to 0, then this stone is destroyed.
+ getter/setter	durability must be in range [0,5]

*3.2.2.3 Class Dynamite /*you must implement from the scratch*/*

This class represents a stone that can be exploded if player dig them

Methods

Method	Description
+ Dynamite(int posX, int posY)	Constructor of this class Set posX, posY
+ void destroy()	Destroy this stone. After that, destroy all the stones around (posX, posY). Hint: can get adjacency stones by calling GameUtilities.getAdjacentStones(posX, posY)

*3.2.2.4 Class WalkingStone /*you must implement from the scratch*/*

This class represents a stone that can walk after a player!

Methods

Method	Description
+ WalkingStone(int posX, int posY)	Constructor of this class Set posX, posY
+ void dig(int digPower)	Dig this stone. Moreover, if the dig power is more than 1, then add score 1 to the game manager
+ void walk()	Command this stone to walk hint: can use GameUtilities.moveWalkingRock (WalkingStone walkingStone) to make this stone walk.

*3.2.2.5 Class Crystal /*Already Provided*/*

This class represents a stone that's harder to destroy than a normal stone and has value.

Fields

Field	Description
- int value	Value of this stone. This value can be as much as they can but can't be less than 1

Methods

Method	Description
+ Crystal(int posX, int posY)	Constructor of this class Set posX, posY Set durability to 5 Set value to 10
+ Crystal(int posX, int posY, int value)	Another constructor of this class Set posX, posY Set durability to 5 Set the value of this stone
+ void dig(int digPower)	Dig this stone by digPower and every time this stone is dug, the game score is increased by 1.

+ void destroy()	When this stone is destroyed all their value is given to the game score
+ getter/setter	value must be equal or more than 1

*3.2.2.6 Class Gear /*you must implement from the scratch*/*

This class represents a stone that's harder to destroy than a normal stone and able to give power to the player.

Fields

Field	Description
- int upgradeValue	Power of this stone can be given to upgrade the player's tool. This value cannot be less than 1.

Methods

Method	Description
+ Gear(int posX, int posY)	Constructor of this class Set posX, posY Set durability to 2 Set upgradeValue to 1
+ Gear(int posX, int posY, int upgradeValue)	Another constructor of this class Set posX, posY Set durability to 2 Set upgradeValue of this stone
+ void destroy()	When this stone is destroyed all their upgradeValue is given to the dig power
+ getter/setter	upgradeValue must be equal or more than 1

3.3 Package Utils /*Already Provided*/

3.3.1 Class GameUtilities

This class contains useful methods that you should use to implement other classes.

Methods

Method	Description
ArrayList<Stone> getAdjacentStones(int x, int y)	Return ArrayList of stones that are adjacent to the given position (x,y).
void removeStone(Stone stone)	Remove stone from the map
+ void moveWalkingRock (WalkingStone walkingStone)	Move walkingStone based on player position

4. Scoring Criteria

The scoring criteria is listed below. There are JUnit files as well as the UML Diagram file. The total score of this section is 16, which will be factored to a net score of 10.

1. **GameManagerTest** **2 points**
    a. digStoneTest 1 point
    b. digDynamiteTest 1 point
2. **HardStoneTest** **3 points**
    a. constructorTest 1 point
    b. durabilityGetterSetterTest 1 point
    c. digTest 1 point
3. **DynamiteTest** **2 points**
    a. constructorTest 1 point
    b. destroyTest 1 point
4. **WalkingStoneTest** **3 points**
    a. constructorTest 1 point

b. digTest 1 point

c. walkTest 1 point

5. **GearTest** **4 points**

   a. noParameterConstructorTest 1 point

   b. parameterConstructorTest 1 point

   c. getterSetterTest 1 point

   d. destroyTest 1 point

6. **UML Diagram of package logic.stone. Save the UML Diagram as umlQ2.png in the root folder of your project.** **2 points**

   a. All classes are present with correct variables and methods 0.5 point

   b. All access modifiers are correct 0.5 point

   c. Inheritance structures are correct 1 point

Export jar (**q2.jar**) file that **includes source code and .class files** and put it in **root** folder of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.