German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Computer Programming Lab, *Spring 2016*
Dragon Ball: **Milestone 1**

Deadline: Fri 4.3.2016 @23:59

This milestone is an *exercise* on the concepts of **object oriented programming (OOP)**. The following sections describe the requirements of the milestone.

By the **end of this milestone**, you should have:

- A packaging hierarchy for your code
- An initial implementation for all the needed data structures
- Basic data loading capabilities form a csv file

1 Build the Project Hierarchy

1.1 Add the packages

Create a new `Java` project and build the following hierarchy of packages:

1. `dragonball.model`
2. `dragonball.model.player`
3. `dragonball.model.character`
4. `dragonball.model.character.fighter`
5. `dragonball.model.world`
6. `dragonball.model.cell`
7. `dragonball.model.battle`
8. `dragonball.model.attack`
9. `dragonball.model.dragon`
10. `dragonball.model.game`
11. `dragonball.controller`
12. `dragonball.view`
13. `dragonball.exceptions`
14. `dragonball.tests`

Afterwards, proceed by implementing the following classes. You are allowed to add more classes, attributes and methods. However, you must use the same names for the provided attributes and methods.

1.2 Naming and privacy conventions

Please note that all your class attributes must be **private**. You should implement the appropriate setters and getters conforming with the access constraints. Please note that getters and setters should match the Java naming conventions. If the instance variable is of type boolean, the getter method name starts by **is** followed by the **exact** name of the instance variable. Otherwise, the method name starts by the verb (get or set) followed by the **exact** name of the instance variable. The first letter of the instance variable should be capitalized. Please note that the method names are case sensitive.

Example 1 *You want a getter for an instance variable called* `milkCount` → *Method name =* `getMilkCount()`

2 Build the (Player) Class

Name : `Player`

Package : `dragonball.model.player`

Type : Class

Description : A class representing a single player playing the game.

2.1 Attributes

All the class attributes are READ and WRITE.

1. `String name`: The player's name.
2. `ArrayList<PlayableFighter> fighters`: Fighters available to the current player.
3. `ArrayList<SuperAttack> superAttacks`: The super attacks learned/unlocked by the player and available for his/her playable fighters.
4. `ArrayList<UltimateAttack> ultimateAttacks`: The ultimate attacks learned/unlocked by the player and available for his/her playable fighters.
5. `int senzuBeans`: Number of Senzu beans collected by the player.
6. `int dragonBalls`: Number of Dragon balls collected by the player.
7. `PlayableFighter activeFighter`: The current active fighter of the player.
8. `int exploredMaps`: The number of maps the player has already explored and won.

2.2 Constructors

1. `public Player(String name)`: Constructor that initializes a `Player` object.
2. `public Player(String name, ArrayList<PlayableFighter> fighters, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks, int senzuBeans, int dragonBalls, PlayableFighter activeFighter, int exploredMaps)`: Constructor that initializes all attributes of the `Player` object. Used when loading the Player from a saved state.

3 Build the (Character) Class

Name : `Character`

Package : `dragonball.model.character`

Type : Class

Description : A class representing a single character belonging to the player. No objects of type `Character` can be instantiated.

3.1 Attributes

All the class attributes are READ only.

1. `String name`: Name of the character

3.2 Constructors

1. `public Character(String name)`: Constructor that initializes a `Character` object.

4 Build the (Fighter) Class

Name : `Fighter`

Package : `dragonball.model.character.fighter`

Type : Class

Description : Subclass of the (`Character`) class representing the fighters in the game. No objects of type `Fighter` can be instantiated. The class implements the `BattleOpponent` interface.

4.1 Attributes

All the class attributes are READ and WRITE.

1. `int level`: The level of the fighter.
2. `int blastDamage`: The value of the blast damage the fighter inflects on a foe during battle.
3. `int physicalDamage`: The value of the physical damage the fighter inflects on a foe during battle.
4. `int healthPoints`: The current health points the fighter has during battle.
5. `int maxHealthPoints`: The max health points the fighter can have during battle. This value is specified according to the `PlayableFighter` subtype, that will be specified in Section 5.
6. `int ki`: The current energy a fighter has during a battle to perform attacks.
7. `int maxKi`: The max energy a fighter can have to perform attacks. This value is specified according to the `PlayableFighter` subtype, that will be specified in Section 5.
8. `int stamina`: The current stamina the fighter has during a battle to block attacks.
9. `int maxStamina`: The max stamina the fighter can have to block attacks. This value is specified according to the `PlayableFighter` subtype, that will be specified in Section 5.
10. `ArrayList<SuperAttack> superAttacks`: List of the super attacks assigned to the Fighter.
11. `ArrayList<UltimateAttack> ultimateAttacks`: List of the ultimate attacks assigned to the Fighter.

4.2 Constructors

1. `public Fighter(String name, int level, int maxHealthPoints, int blastDamage, int physicalDamage, int maxKi, int maxStamina, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks)`: Constructor that initializes all attributes of the `Fighter` object.

5 PlayableFighter

Name : `PlayableFighter`

Package : `dragonball.model.character.fighter`

Type : Class

Description : Subclass of the (`Fighter`) class representing fighters that the player can choose to play with. No objects of type `PlayableFighter` can be instantiated. The class implements the `PlayableCharacter` interface.

5.1 Attributes

All the class attributes are READ and WRITE.

1. `int xp`: The current experience points the fighter has.
2. `int targetXp`: The target experience points the fighter needs to reach the next level.
3. `int abilityPoints`: The number of ability points the fighter has.

5.2 Constructors

1. `public PlayableFighter(String name, int level, int xp, int targetXp, int maxHealthPoints, int blastDamage, int physicalDamage, int abilityPoints, int maxKi, int maxStamina, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks)`: Constructor that initializes all attributes of the `PlayableFighter` object. Used when loading the Player from a saved state. Your constructor must utilize the constructor of the (`Fighter`) class. Any fighter should have his/her `currentHealthPoints` and `currentStamina` set to their maximum value and the `currentKi` to 0.
2. `public PlayableFighter(String name, int maxHealthPoints, int blastDamage, int physicalDamage, int maxKi, int maxStamina, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks)`: Constructor that initializes a `PlayableFighter` object. Your constructor should utilize the above `PlayableFighter` constructor. Any fighter should have his/her `currentHealthPoints` and `currentStamina` set to their maximum value and the `currentKi` to 0.

   The default values of the remaining attributes should be passed as follows:

   - **level**: 1
   - **xp**: 0
   - **targetXp**: 10
   - **abilityPoints**: 0
3. `Subclasses:` Each of the subclasses representing the different playable fighter types should have its own constructor that utilizes the `PlayableFighter` constructor.

There are 5 different different types/races of playable fighters available in the game. Each playable fighter type is modelled as a subclass of the (`PlayableFighter`) class. Each playable fighter should be implemented in a separate class within the same package as the `PlayableFighter` class. Carefully consider the design of the constructors of each subclass.
The following table gives their types and the max values of their attributes:

Class name	Blast Damage	Physical Damage	Max Health Points	Max Ki	Max Stamina
`Earthling`	50	50	1250	4	4
`Saiyan`	150	100	1000	5	3
`Namekian`	0	50	1350	3	5
`Frieza`	75	75	1100	4	4
`Majin`	50	50	1500	3	6

All playable fighters only have the attributes they inherit the attributes from the `Fighter` class.
Only Saiyans have an additional attribute:

1. `boolean transformed`: Indicates whether this Saiyan fighter is currently transformed or not during a battle. This attribute is READ and WRITE.

6 Build the (NonPlayableFighter) Class

Name : `NonPlayableFighter`

Package : `dragonball.model.character.fighter`

Type : Class

Description : Subclass of the `Fighter` class representing fighters that cannot be chosen by the player but rather act as foes that the player encounters and battles in the world. The class implements the `NonPlayableCharacter` interface.

6.1 Attributes

All the class attributes are READ only.

1. `boolean strong`: Indicates whether this non playable fighter represents a weak or a strong foe (Boss). If the value is true then the foe is strong.

6.2 Constructors

1. `public NonPlayableFighter(String name, int level,int maxHealthPoints, int blastDamage, int physicalDamage, int maxKi, int maxStamina, boolean strong, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks)`: Constructor that initializes a `NonPlayableFighter` object. Your constructor must utilize the constructor of the (`Fighter`) class.

7 PlayableCharacter

Name : `PlayableCharacter`

Package : `dragonball.model.character`

Type : Interface

Description : Interface containing the methods available to playable characters

8 NonPlayableCharacter

Name : `NonPlayableCharacter`

Package : `dragonball.model.character`

Type : Interface

Description : Interface containing the methods available to non-playable characters

9 BattleOpponent

Name : `BattleOpponent`

Package : `dragonball.model.battle`

Type : Interface

Description : Interface containing the methods available to fighters within battle while acting against an opponent.

10 Build the (World) Class

Name : `World`

Package : `dragonball.model.world`

Type : Class

Description : A class representing the world that the player navigate throughtout gameplay.

10.1 Attributes

All the class attributes are READ only.

1. `Cell[][] map`: A randomly generated 2D array list of `Cell` objects representing the player's current map in the world.
2. `int playerColumn`: Specified the x-coordinate of the Player's position in the map.
3. `int playerRow`: Specified the y-coordinate of the Player's position in the map.

10.2 Constructors

1. `public World()`: Constructor that initializes a `World` object to have a 10x10 2D array map.

10.3 Methods

1. `public void generateMap(ArrayList<NonPlayableFighter> weakFoes, ArrayList<NonPlayableFighter> strongFoes)`: It should generate the whole map grid randomly. It should be filled with:
   - **1 Boss**: A random boss should be positioned in the top left of the map i.e. cell (0,0).
   - **15 Weak Foes**: The weak foes should be randomly positioned on the map. The weak foes should be chosen randomly from the input `ArrayList<NonPlayableFighter> weakFoes`. The weak foes apprearing in each map could be duplicates.
   - **3-5 Senzu Beans**: Randomly choose how many Senzu beans should appear in each map (Range:3-5). The senzu beans should be randomly positioned in the map.

- **1 Dragon Ball**: The dragon ball should be randomly positioned in the map.
- **Player**: The bottom right of the map i.e. cell (9,9) should be kept empty and not have any item randomly assigned to it as the player will be later positioned there.

All the remaining cells should be empty and no cell could have two of the above at the same time.

2. `public String toString()`: It should output the cells of the map representing the world in a readable form. This method should utilize the `toString` methods of the `Cell` subclasses.

11 Cell

Name : `Cell`

Package : `dragonball.model.cell`

Type : Class

Description : A superclass representing a single cell in the map that a character can pass through. No objects of type `Cell` can be instantiated.

11.1 Methods

`public String toString()`: Displays contents of a `Cell` in a readable way. This method should be overridden in all subclasses of `Cell`.

1. **Empty Cell**: Should be represented as an empty square bracket with a space inside "`[ ]`".
2. **Foe Cell**: Should be represented as a square bracket "`[]`" containing a letter representing whether this cell contains a weak or a strong foe (Boss). So cells with weak foe fighters are represented as "`[w]`" while the others are represented as "`[b]`".
3. **Collectible Cell**: Should be represented as a square bracket "`[]`" containing a letter representing whether this cell contains a Senzu bean or a Dragon ball. So cells with Senzu beans are represented as "`[s]`" while the others are represented as "`[d]`".

There are 3 different cell types available in the game. Each cell type is modelled as a subclass of the (`Cell`) class. Each cell should be implemented in a separate class within the same package as the `Cell` class. Carefully consider the design of the constructors of each subclass.
The different cell types are:

1. **Empty Cell**: Empty cells contain nothing. The player just passes through them and moves to the next cell. (**Class name:** `EmptyCell`)
2. **Foe Cell**: Fighter cells contain either strong or weak foes. These are the players gateway to battle with the foe. (**Class name:** `FoeCell`)

   Attributes

   (a) `NonPlayableFighter foe`: The foe that the current fighter will battle. This attribute is READ only.
3. **Collectible Cell**: Collectible cells contain an item that can be collected by the player. These can either be dragon balls or senzu beans. (**Class name:** `CollectibleCell`)

   Class attributes

   (a) `Collectible collectible`: Item that can be collected by the player. This attribute is READ only.

12 Collectible

Name : `Collectible`

Package : `dragonball.model.cell`

Type : Enum

Description : An enum representing the different types of the collectibles. Possible values are: SENZU_BEAN and DRAGON_BALL.

13 Build the (Battle) Class

Name : `Battle`

Package : `dragonball.model.battle`

Type : Class

Description : A class representing a single battle that takes place throughout the gameplay.

13.1 Attributes

All the class attributes are READ only.

1. `BattleOpponent me`: Represents the active fighter of the player which will take part in the battle.
2. `BattleOpponent foe`: Represents the foe the fighter is currently batteling
3. `BattleOpponent currentOpponent`: represents the the fighter in turn.

13.2 Constructors

1. `public Battle(BattleOpponent me, BattleOpponent foe)`: Constructor that initializes a `Battle` object. Both fighters should have their `currentHealthPoints` and `currentStamina` set to their maximum value and their `currentKi` to 0.

14 Attack

Name : `Attack`

Package : `dragonball.model.attack`

Type : Class

Description : A superclass representing an attack that can be performed during battle. No objects of type `Attack` can be instantiated.

14.1 Attributes

All the class attributes are READ only.

1. `String name`: The name of the `Attack`.
2. `int damage`: Represents the damage inflicted on the opponent upon performing the attack.

14.2 Constructors

1. `public Attack(String name, int damage)`: Constructor that initializes an `Attack` object.

There are 3 different types of attacks available in the game. Each attack type is modelled as a subclass of the (`Attack`) class. Each attack should be implemented in a separate class. The different attack types are:

1. **Physical Attack**: Normal attack with damage 50. The name of this attack should be "Physical Attack". (**Class name:** `PhysicalAttack`)
2. **Super Attack**: There are different super attack types. Each super attack has its specific damage value. The super attacks and their damage values are read from csv files in the `Game` class. (**Class name:** `SuperAttack`)

   Subclasses

   (a) `MaximumCharge`: Special type of `SuperAttack`. The damage of `MaximumCharge` is 0. The name of this attack should be "Maximum Charge".
3. **Ultimate Attack**: There are different ultimate attack types. Each ultimate attack has its specific damage value. The ultimate attacks and their damage values are read from csv files in the `Game` class. (**Class name:** `UltimateAttack`)

   Subclasses

   (a) `SuperSaiyan`: Special type of `UltimateAttack`.The name of this attack should be "Super Saiyan". The damage of `SuperSaiyan` is 0.

15 Build the (Dragon) Class

Name : `Dragon`

Package : `dragonball.model.dragon`

Type : Class

Description : A class representing the dragon encountered by the player upon having collected 7 dragon balls.

15.1 Attributes

All the class attributes are READ only.

1. `String name`: The name of the dragon.
2. `ArrayList<SuperAttack> superAttacks`: List of super attacks the player can choose one from as a wish from the dragon.
3. `ArrayList<UltimateAttack> ultimateAttacks`: List of ultimate attacks the player can choose one from as a wish from the dragon.
4. `int senzuBeans`: Number of senzu beans the player can wish for from the dragon.
5. `int abilityPoints`: Number of ability points the player can wish for from the dragon.

15.2 Constructors

1. `public Dragon(String name, ArrayList<SuperAttack> superAttacks, ArrayList<UltimateAttack> ultimateAttacks, int senzuBeans, int abilityPoints)`: Constructor that initializes a `Dragon` object. .

16 Build the (Game) Class

Name : `Game`

Package : `dragonball.model.game`

Type : Class

Description : Class representing the whole game play.

16.1 Attributes

All the class attributes are READ only.

1. `Player player`: The current player of the game.
2. `World world`: The player's current world i.e. map.
3. `ArrayList<NonPlayableFighter> weakFoes`: List of weak foes available for current player in current world. The list is loaded from the provided CSV file. It should only be loaded upon initialization of a new map.
4. `ArrayList<NonPlayableFighter> strongFoes`: List of strong foes available for current player in current world. The list is loaded from the provided CSV file. It should only be loaded upon initialization of a new map.
5. `ArrayList<Attack> attacks`: List of attacks available for current player. The list is loaded from the provided CSV file. It should only be loaded once on game start.
6. `ArrayList<Dragon> dragons`: List of dragons available for current player in current world. The list is loaded from the provided CSV file. It should only be loaded once on game start.

16.2 Constructors

1. `public Game()`: Empty constructor that initializes a `Game` object. In this constructor, the correct methods should be invoked to load the CSV files and generate the `World` map.

16.3 Methods

1. `loadCSV(String filePath)`: A private helper method that reads the CSV file with `filePath` and stores its contents in any data type of your choice. You can also decide on the return type of the method. Make sure that any error in loading attacks, foes or dragons could originate from this method.
2. `private void loadAttacks(String filePath)`: Reads the CSV file with `filePath` and loads the attacks into the `attacks` ArrayList. Your method should work on the output of the `loadCSV(String filePath)` method.
3. `private void loadFoes(String filePath)`: Reads the CSV file with `filePath` and loads the non-playable fighters into the `weakFoes` and `strongFoes` ArrayLists. Your method should work on the output of the `loadCSV(String filePath)` method.
4. `private void loadDragons(String filePath)`: Reads the CSV file with `filePath` and loads the dragons into the `dragons` ArrayList. Your method should work on the output of the `loadCSV(String filePath)` method.

16.4 Description of CSV files format

1. **Foes**
   (a) Each foe will be described using 3 consecutive lines.
   (b) The first line represents the foe's data as follows: **Name, Level, MaxHealth, BlastDamage, PhysicalDamage, MaxKi, MaxStamina, a boolean** representing whether this foe is a Boss (strong foe (TRUE) or weak foe (FALSE)).
   (c) The second line represents the names of the super attacks of this foe. The name has to be exactly as it is in the csv file. Up to 4 super attacks separated by commas. In case of no super attacks it will be an empty line "\n".
   (d) The third line represents the names of the ultimate attacks of this foe. Up to 2 ultimate attacks separated by commas. In case of no ultimate attacks it will be an empty line "\n".
   (e) The data has no header, i.e. the first line represents the first foe.
   (f) The parameters are separated by a comma (,).
   (g) The data of the foe Fighters is found in `Database-Foes-Range1.csv`.
   (h) The `Range1` suffix will take different numbers in the coming milestones. For the purposes of this milestone you will only use the `Database-Foes-Range1.csv` file.

   Example 2 *An entry for a **Foe** will look like:*
   Goku,10,3000,350,400,5,6,TRUE
   Kamehameha,Maximum Charge
   Super Kamehameha,Spirit Bomb

2. **Attacks**
   (a) Each line represents an attack.
   (b) The data has no header, i.e. the first line represents the first attack.
   (c) The parameters are separated by a comma (,).
   (d) The data of an **attack** is added in the file as follows: **Type, Name, Damage**. The type indicates whether it is a super (SA), ultimate (UA), maximum charge (MC) or super saiyan (SS) attack.
   (e) The data of the Attacks is found in `Database-Attacks.csv`.

   Example 3 *An entry for an **Attack** will look like:*
   SA,Kamehameha,250

3. **Dragons**
   (a) Each dragon is described in 3 lines
   (b) The first line represents the Dragon's data as follows: **Name, Number of senzu beans** this dragon will grant upon wishing and the **Number of ability points** this dragon will grant upon wishing.
   (c) The second line represents the list of names of the super attacks that will be granted from upon wishing.
   (d) The third line represents the list of names of the ultimate attacks that will be granted from upon wishing.
   (e) The data has no header, i.e. the first line represents the first dragon.
   (f) The parameters are separated by a comma (,).
   (g) The data of the Dragons is found in `Database-Dragons.csv`.

   Example 4 *An entry for a **Dragon** will look like:*
   Shenron,10,5
   Big Bang Kamehameha,Emperor's Death Beam,Demon Wave,Guilty Flash
   Super Big Bang Kamehameha,Final Shine Attack,Final Galick Gun,Explosive Demon Wave

Hint: When loading foes and dragons, you should think about how to retrieve their attacks from the preloaded list of attacks.