Computer Science AP
Abstract Classes Worksheet

These are questions associated with each video in the Abstract Classes Section.

Questions for Video: Inheritance and Abstract Classes

What is an Abstract class?
A class that has methods that are not implemented (coded). Must be labeled 'abstract'.

What is an abstract method and what does it look like when declared in an abstract class?
Has the keyword 'abstract' and has no code.
public abstract void talk();

An abstract class called *SoundManager* exists in a student project and has a default constructor with no parameters. Can the following line of code be used in a runner program?
SoundManager mySoundManager=new SoundManager()
No, you cannot instantiate an instance of an abstract class – it's code is not complete yet!

In the InhertianceExamples01 project, StudentTeacher02 package, we made the Student class an abstract class. What arguments could you use to support the decision to use an abstract class in this project?
I want every Student to have the methods talk, getAverage, and getMark but I also know that each subclass of Student will code these differently. This is the perfect reason to make the methods abstract. Let the subclasses inherit all the regular methods that are coded and let them code their own version of talk, getAverage, and getMark.

Consider the classes

```
public abstract class A {
    public abstract void stuff()
    public abstract int add(int a, int b)
    public void clear() { //coded but not shown here }
    public void undo() { //coded but not shown here }
}

public class B extends A {


}
```

To complete class B, describe in words (don't code it) what the programmer would have to include inside of class B.
You would have to code the abstract methods stuff and add.

This question is going to ask you to design a pattern of inheritance using classes and abstract classes for a project. So you are trying to design something that looks likes the diagrams with the boxes and the arrows that you saw in the videos. Decide on what you think would be good relationships between the classes if you were going to code this project. Add in some of the key methods (the ones listed below)

The project is a video game that invovles the following classes:
GamePiece, Player, Enemy, MovingEnemy, Bases, Tanks, Planes.

Assorted information about the behavior of the game:

- A gamepiece is anything that is going to be shown in the game and drawn on the screeen. This includes the player, enemies, bases, tanks, planes, etc.
- Everything that is to be drawn on the screen needs a draw() method
- Everything needs a setPosition(int x, int y) method so they can be placed in the proper place when the game starts.
- A Base represents an enemy that is a building that will sometimes fire at the player. Bases never move.
- A Tank is an enemy and moves and fires strong shots at the player
- A Plane is an enemy and moves and fires lots of lasers at the player. It can also bomb the player.
- A MovingEnemy should be able to moveForward, turnLeft, turnRight, and stop. Tanks and Planes are moving enemies but will have different implementations of moveForward, turnLeft, turnRight, and stop since they are very different vehicles.
- All enemies have a variable that keeps track of their life and a methods that allow the enemy to takeDamage and checkIfDead (based on the life variable).
- Players have a variable that keeps track of life and a methods that allow the player to takeDamage and checkIfDead (based on the life variable).

Keep in mind that while there are many 'really wrong' ways to set this project up, there can be more than one good way – so if you sketch up a few different ways that seem to have the same number of pro's and con's, just go with one of them!

Sample of what you are trying to do:

public Class A
variable num, thing
methodA()
methodA2()

public ClassB extends A
methodA()
methodB()

public ClassC extends A
variable x
variable value
methodA()
methodC()

public abstract class D
variable dd //use the class names provided, some of the variables listed,
abstract methodD() //some of the methods listed, and create an efficient
abstract mehodD2()

One way could be

abstract class GamePiece
  int x, y;
  int life;
  abstract draw()
  setPosition() coded
  takeDamage() coded
  checkIfDead() coded

abstract class Enemy
  abstract fire()

abstract class MovingEnemy extends Enemy
  abstract moveFoward()
  abstract moveLeft()
  abstract moveRight()

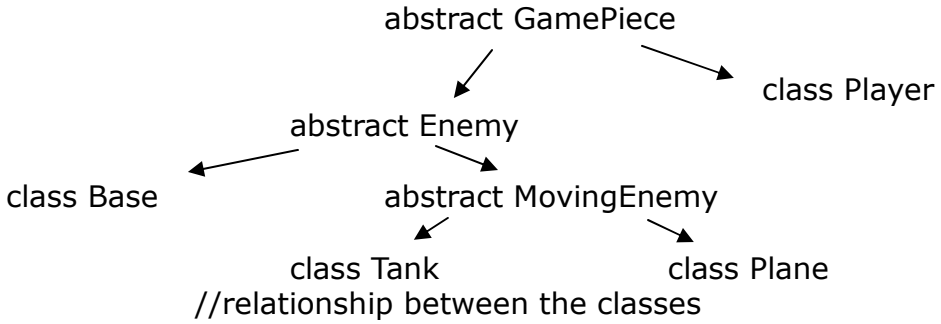
class Tank extends MovingEnemy
  fire() coded
  draw() coded
  moveFoward() coded
  moveLeft() coded
  moveRight() coded

class Plane extends MovingEnemy
  fire() coded
  draw() coded
  moveForward() coded
  moveLeft() coded
  moveRight() coded

class Base extends Enemy
  fire() coded
  draw() coded

class Player extends GamePiece
  draw() coded

Here's a little diagram showing the pattern of inheritance. Notice that each level of inheritance should add some functionality to the class that it extending.



//relationship between the classes