Question 1:

Given an abstract base class Pet.

<<Abstract>> Pet
-name: String -breed:String -weight:double
+Pet(name: String, breed:String, weight:double) +setName(String name) +getName():name +getWeight():double +display() +*sound()* +*eat()*

Note: *sound() and eat()* are abstract method()
And two child class Dog and Cat that extends Pet

Dog
-microchipped:boolean
+Dog(name: String, breed:String, weight:double, microchipped:boolean) +sound() +eat(String food) +eat() +display()

Cat
+Cat(name: String, breed:String, weight:double)
+sound() +eat() +display()

a. Implement the above three classes.
   - Sound => Cat meow and Dog bark.
   - Eat => Cat eat fish and Dog eat cookies or other food that it is given.
b. Create a few Cat and Dog objects and put into an ArrayList.
c. Sound and feed the five pets.
d. If a dog object is assigned to a Pet datatype, can we feed the dog with food?
e. Display the following data:
   - name of lightest pet,
   - name of heaviest pet, and
   - average weight of the five pets.

Question 2

The Java game program Attack allows two players to choose an avatar (Ranger or Digger) each and attack each other for five rounds.
When the game starts, each avatar will generate a power value based on certain algorithm. The power value will then be used to attack the opponent. After five rounds of attack, the game will end and display the winner.

The display below shows two sample runs of the game.
Text in **bold** is the input from keyboard by user.

```
Game started
Player 1 enter 1 for Ranger, 2 for Digger:1
Player 2 enter 1 for Ranger, 2 for Digger:2
Round:1
Round:2
Round:3
Round:4
Round:5
Player 1 wins!
```

```
Game started
Player 1 enter 1 for Ranger, 2 for Digger:2
Player 2 enter 1 for Ranger, 2 for Digger:1
Round:1
Round:2
Round:3
Round:4
Round:5
Player 2 wins!
```

Refer to the program listing of the Attack game on the next few pages.

What if we have more than 2 avatars in the game? The possible combinations becomes un-manageable!

The "game" engine codes should cater for many types of avatars without any changes.

Add in two more avatars to the game.

- Black Spider
- Monster
- Make up the attack algorithm

Apply the Object-Oriented concepts of Inheritance, Override, Interface and Polymorphism to rewrite the entire game program such that it is scalable and extensible.

The Java program below shows the implementation of the game “Attack”.

```
public class AttackGame {

	public static void main(String[] args) {
		new AttackGame().run();
	}

	public void run() {
		//Player choose avatar
		println("Game started");
		int avatar1 = Keyboard.readInt("Player 1 enter 1 for Ranger, 2 for Digger:");
		int avatar2 = Keyboard.readInt("Player 2 enter 1 for Ranger, 2 for Digger:");

		//Game engine
		int force1=0,force2=0,power1=0,power2=0;

		if (avatar1 == 1 && avatar2 == 1) {

			Ranger player1 = new Ranger(); Ranger player2 = new Ranger();
			for(int round=1;round<=5;round++){
				println("Round:"+round);
				force1 = player1.attack();
				force2 = player2.attack();
				player1.lostPower(force2);
				player2.lostPower(force1);
				power1 = player1.getPower();
				power2 = player2.getPower();
			}

		} else if (avatar1 == 1 && avatar2 == 2) {

			Ranger player1 = new Ranger(); Digger player2 = new Digger();

			for(int round=1;round<=5;round++){
				println("Round:"+round);
				force1 = player1.attack();
				force2 = player2.attack();
				player1.lostPower(force2);
				player2.lostPower(force1);
				power1 = player1.getPower();
				power2 = player2.getPower();
			}

		} else if (avatar1 == 2 && avatar2 == 1) {

			Digger player1 = new Digger(); Ranger player2 = new Ranger();

			for(int round=1;round<=5;round++){
				println("Round:"+round);
				force1 = player1.attack();
				force2 = player2.attack();
				player1.lostPower(force2);
				player2.lostPower(force1);
				power1 = player1.getPower();
				power2 = player2.getPower();
			}

		}else if (avatar1 == 2 && avatar2 == 2) {

			Digger player1 = new Digger(); Digger player2 = new Digger();
			for(int round=1;round<=5;round++){
				println("Round:"+round);
				force1 = player1.attack();
```

```
                        force2 = player2.attack();
                        player1.lostPower(force2);
                        player2.lostPower(force1);
                        power1 = player1.getPower();
                        power2 = player2.getPower();
                  }
            }

            //Determine Winner
            if(power1>power2){
                  println("Player 1 wins!");
            }else if(power1<power2){
                  println("Player 2 wins!");
            }else{
                  println("It is a tie!");
            }
      }

      public static void println(String s) {
            System.out.println(s);
      }
}
```

```
import java.util.Random;

public class Digger {

      private int power, factor;
      private Random r = new Random();

      public Digger(){
            this.power = r.nextInt(5);
            this.factor = r.nextInt(10);
      }

      public int attack(){

            //Algorithm for attack method in Digger
            int force = factor*r.nextInt(Math.abs(power)+1);
            return force;
      }

      public void lostPower(int p){

            power-=factor*p;
      }

      public int getPower(){

            return power;
      }
}
```

```
import java.util.Random;

public class Ranger {

	private int power, factor;
	private Random r = new Random();

	public Ranger(){
		this.power = r.nextInt(10);
		this.factor = r.nextInt(5);
	}

	public int attack(){
		//Algorithm for attack method in Ranger
		int rNum = r.nextInt(1);
		int force = factor*r.nextInt(Math.abs(power)+1);

		if(rNum%2==0) force += r.nextInt(Math.abs(power/(factor+1)+1));
		else force -= r.nextInt(Math.abs(power/(factor+1)+1));

		return force;
	}

	public void lostPower(int p){
		power-=factor*p;
	}

	public int getPower(){
		return power;
	}
}
```