Aleksandr Petrosyan OOP Supervision Work 3

----------**Question** 1

Java Generics as opposed to C++ Templates doesn't create multiple classes by just substitution of the parameter. Instead it makes use of the fact that every object in java inherits from the super-class Object. Initially when there were no generic types, Java only used objects for this purpose, defining the most general form of object manipulation within generic types and classes, and only in some cases specifying what that is. Since this used to cause many problems, and the Java developers didn't want to break reverse compatibility they couldn't implement generics exactly as they are implemented in C++, and instead of defining multiple classes, just do a little more type checking with objects.

Since primitives are NOT Objects, they cannot be manipulated in the same way and therefore Generics can't work with them. However they can work with the boxed versions of primitives, for example with Integers rather than ints.

----------**Question** 2

The First Naive Attempt is to create a class that extends both the abstract class AbstractList and the List interface. Eclipse Kindly does fill in most of the methods in, however it doesn't automatically assign a generic type to the class in question - CollectionArrayList, which can be done manually to avoid the ambiguity of defining <E> inside the class itself.

Tried creating a generic array which didn't work. Tried creating an array of objects and casting them to the generic Type. Th8is should work in all cases, but the compiler duly warns of the possible problem of this being unchecked. The only way oif creating anything

//----------
CollectionArrayList.java
//----------

```
package uk.ac.cam.ap886.oopjava.supervision3;

import java.util.AbstractList;
import java.util.List;

public class CollectionArrayList<E> extends AbstractList<E> implements List<E> {

	protected E[] list;
	protected int last;
	private final int INITIAL_ARRAY_LENGTH = 50;

	@SuppressWarnings("unchecked")   //It is checked we know this going to be an E array
	public CollectionArrayList (){
		list = (E[]) new Object[INITIAL_ARRAY_LENGTH];
		last = 0;
	}

	@SuppressWarnings("unchecked")
	public CollectionArrayList (int length){
		list = (E[]) new Object[length];
		last = 0;
```

```
	}

	@Override
	public E get(int index) {
		if(index <last && index>=0){
			return list [index];
		}
		else{
			throw new IndexOutOfBoundsException
			("Index "+new Integer(index).toString()+" Not present");
		}
		//To be Collections Compliant we need to throw specific exceptions
		//otherwise there's no sense in implementing the List interface.
	}

	@Override
	public E remove(int index) {
		if(last ==0){
			throw new IndexOutOfBoundsException("Tried to remove from empty
List");
		}
		E removedElement = get(index);
		//Avoids duplication of catching, also
		//Helps if we want to change the numbering convention
		//To or from zero based;
		for(int i=index;i<last;i++){
			if(i + 1 >= last){//preLast element. Just need to shorten the array
				last--;
				break;
			}else{			//Otherwise need to shift everything left
				list[i] = list[i+1];
			}
		}
		return removedElement;
	}

	@SuppressWarnings("unchecked")//Same as in the Constructor
	@Override
	public E set(int index, E newValue) {
		E cache = null;
		if(index <0){
			throw new IndexOutOfBoundsException
			(new Integer(index).toString()+ " is not a valid positive integer");
		}else if(index >last+1){
			throw new IndexOutOfBoundsException
			(new Integer(index).toString()+ " Exceeeded "+ new
Integer(last+1).toString());
		}else {
			cache = get(index);
			if(index <=last){
				list[index] = newValue;
			}else{  //Setting exactly last element
```

```
                        if(index <=list.length ){
                                last = index;
                                list[index] = newValue;
                        }else{//Need to resize the array
                                E[] oldList = list;
                                list = (E[]) new Object[oldList.length +
INITIAL_ARRAY_LENGTH];
                                for(int i=0;i<oldList.length;i++){
                                        list[i] = oldList[i];
                                }
                                last = index;
                                list[index]= newValue;
                        }
                    }
                }
                return cache;
        }
        @Override
        public boolean add(E newElement){
                set(last+1, newElement);
                return true;

        }

        @Override
        public int size() {
                return last;
        }

}
```

----------Question 3

The last three lines of code are internally optimised by the compiler. Instead creating separate objects for the same (Immutable) string constant, it jsut sets both references to the same address, thus coincidentally in the last example both strings are not only equal not only in the lexicographical sense, but also represent the same memory cell.

In the first example, however, we explicitly specify creationg of new string objects, which by all means need to be distinct. Although they are the same in the lexicographical sense, the references s1 and s2 point to different addresses.

----------Question 5

To be comparable a class needs to implement the java standard class comparable, with the generic type equal to what we want tocompare it to. In this case we want to compare cars to cars, thus the generic type is specified to be car. Next, we neeed to check in order how we want to sort things. e.g. at first we want to sort by manufaturer lexicographically, and only then sort by age. Although in this case all of it could be done in one return statement with a ternary operator, since it is possible to extend the class of cars and add new fields and then the usetr might want to re-implement sorting doing this with if statements is much easier and understandable.

----------**Question** 6

Both design patterns are considered to be part of the behavioural design patterns. These are considerably similar but not identical.

The State pattern alters the behaviour of methods based on internal state, which in turn gets altered by the user input. This is very much common in all imperative languages, as most functions that need to chekc a datastructure like a list for presence of at least one something intrinsically need to be terminated by changing some external state. This is efficient in terms of runtime performance somewhat inefficient in terms of the complexity of the written code.

This design pattern is primarily used for designing serial modification interfaces, for example wizards and dialogue boxes.

This is implemented by first creating an intwerface for a task, then designing multiple implementations that correspond to different states, and at last by creating a context class that has a variable of the interface type and depending on the user input changes the state from on to the other.

For example This code will change state after each output and will print sad and happy greeting every other time.

```
//
SateContext.java
//

package uk.ac.cam.ap886.oopjava.supervision3;

public class StateContext {
	private StateGreet mState;

	public StateContext (){
		mState = new StateGreetHappy();
	}
	public void setState(StateGreet newState){
		mState = newState;
	}
	public void greet(String name){
		System.out.print("Hey, "+ name + ". ");
		mState.greet(this);
		System.out.format("%n%n%n");

	}
}
```

```
//
StateGreet.java
//
package uk.ac.cam.ap886.oopjava.supervision3;
```

```
public interface StateGreet {
	public void greet (StateContext context);

}
```

//
StateGreetSad
//

```
package uk.ac.cam.ap886.oopjava.supervision3;

public class StateGreetActual {
	public static void main(String[] args){
		StateContext sc = new StateContext();

		for(int i=0;i<5;i++){
			sc.greet("Stephen");
		}
	}
}
```

//
StateGreetHappy
//

```
package uk.ac.cam.ap886.oopjava.supervision3;

public class StateGreetHappy implements StateGreet {


	int mood;
	@Override
	public void greet(StateContext context) {
		System.out.println("Hello, how are you doing? ");
		context.setState(new StateGreetSad());
	}

}
```

//
StateGreetSad
//

```
package uk.ac.cam.ap886.oopjava.supervision3;

public class StateGreetSad implements StateGreet {

	@Override
	public void greet(StateContext context) {
		System.out.println("Thanks for noticing me. ");
```

```
                context.setState(new StateGreetHappy());
        }

}
```

The Strategy pattern is similar to the state pattern in terms that in this case the behaviour is determined at runtime and it uses several encapsulations and relies heavily on Dynamic polymorphism.

THe most classical example is creating a calculator class. Where the user input determines both the nnumbers and the type of operation to be performed. They are interchanged because instead of having multiple if statements we just have an operation interface and multipole implementaitions of it (e.g. addition subtraction etc.).

THere are several minor differences concerning the implementations of both the state and strategy pattern
1) States have a reference to the context object taht contains them, while stategies usually don't. This could be a bonus to the state pattern as although it is more complicated to implement, when something goes wrong all taht information can be easily retrieved and the bug can be easily localised.
2) States are allowed to themselves change the context's state. In the above example states have decided to change the state of the context from happy to sad and vice versa every other time. Strategies aren't typically allowed to do that and this gives more freedom to decide how to change the context inside the very program. On the other hand this also negate some of the advantages of modularity because more and more code that describes this behaviour has to be put in the client class.
3) States are contained within and are created by the context, while Strategies are passed as parameters. This is a more functional approach and it has its benefits in terms of that if the context doesn't depend on any other external machine state (which it shouldn't in a strategy design pattern) its behaviour is entirely predictable and Lambda-Calculus compliant.
4) Strategies handle a specific task, while the State handles almost everything the object does.

However it's virtually impossible to implement anything following only state or only strategy like pattern, the real implementation will always be a mixture of the two providing some middle ground between simplicity and stability.

----------**Question** 6

This particular pattern is used to refrain from implementing all possible combinations of a specific task, but rather allow the programmer or the user to compose what they need by creating a an object of the instance of decorators.

For example let's say taht we want to input the first line of a gzip compressed file to the system. To do this correctly we will nbeed to create an objet of the type reader that is exactly a spcific implementation of a FileReader GzipReader and a BufferedReader. to do this we could do the following.

```
InputStream fileStream = new FileInputStream(filename);  //Read From File
InputStream gzipStream = new GZIPInputStream(fileStream);       //Create a Gzip Stream e.g.
Reader decoder = new InputStreamReader(gzipStream, "UTF-8");//Decompress
BufferedReader buffered = new BufferedReader(decoder);            //Pass the decompressed file to
```

the reader.

----------**Question** 7

The Singleton pattern is a pattern taht disallows instantiating a class more than once. THis could be done eagerly or lazily, (e.g. without the user specifying that needs an object and onl when te user specifies).

TO implement it we typically need a class of the type, with a private constructor and a field of the same type, then to be able to make use of it we need a static method that returns the reference to the field and implicitly calls the constructor. This will result in the fact that there can only be one instance of a class at a time.

```
package uk.ac.cam.ap886.oopjava.supervision3;

public class Singleton {
	private static final Singleton INSTANCE = new Singleton();

	private Singleton (){
		//Can't do anything at this point, because there's only one field;
	}

	public static Singleton getInstance(){
		return INSTANCE;
	}
}
```

The Above example makes use of eager instantiation, i.e. it creates an instance regardless of whether it's actually ever been called.

```
package uk.ac.cam.ap886.oopjava.supervision3;

public class LazySingleton {
	private static LazySingleton instance;
	private LazySingleton(){}

	public static LazySingleton getInstance(){
		if (instance == null){
			instance = new LazySingleton();
		}
			return instance;
	}
}
```

The Second code example features lazy instantiation that instantiates the class only when it's needed if it's needed.

// ----------

The singleton pattern has a limited set of applications. FIrst and foremost it's primary purpose is to prevent more than one instance of objects running at one time.

1)This could be useful in program design. We would like to have a large program like a music player not to open a new window every time we try to play a song.One of the ways of achieveing this is making the program a singleton for the operating system.
2)This could also be useful if we want to lock out modification to a particular file. For example this is used in the visudo, a modification of vi designed to lock out all modification of the sudoers file.

Among the shortcomings is:
1) Having only one instance of a class is against the object oriented paradigm. We might have also made everything static, with almost the same effect.
2) If two threads runa function that does something to a singleton, the operations may only be done serially, which intorduces problems both with scheduling and concurrency. Additionally if the methods called by the threads alter then internal state of the singleton, this may create unpredictable results.

----------**Question** 8

Among the design patterns least of all were covered creation pattterns. One of them is the Builder pattern, thaat counteracts the anti-pattern of telescoping constructors. WHen the created object has far too many fields to be initialised and some of them might be initialised indepenently, it often becomes necessary to an ojbect - builder and feed it as the single argument to the non-default constructor. The reason for doing this is avoiding the duplicated code for every possible combination of initialisation parameters that might neven not be known at compile time. Thisis also signals that the amount of fields or the class in question doesn't necessarily reflect the whole essence of the concept (otherwise, all fields had to have been initialised every single time.

This is very well applied to the problem scenarios where several supplied parameters are optional for exmaple when we consider a person in the medieval England, he could have a number of titles and lands.

```
//
MedievalBuilderExmaple.java
//

package uk.ac.cam.ap886.oopjava.supervision3;

public class MedievalBuilderExample {
	//Person - specific
	int age=0;
	String name=null;
	String Surname=null;

	//Optional parameters
	String dukeOf=null;
	String viscountOf=null;
	String lordName=null;
	String lordOf=null;
	boolean gender=false;
	String knownFor=null;

	public MedievalBuilderExample(MedievalBuilder capsule){
		//Should always find something to initialise with
```

```
		age = capsule.getAge();
		name = capsule.getName()==null?"Anonymous":capsule.getName();
		Surname = capsule.getSurname()==null?"Unfamiliate":capsule.getSurname();


		//Don't really have to be initialised
		dukeOf =		capsule.getDukeOf();
		viscountOf =	capsule.getViscountOf();
		lordName =		capsule.getLordName();
		lordOf =		capsule.getLordOf();
		gender =		capsule.isGender();
		knownFor =		capsule.getKnownFor();
	}

	public void prettyPrint(){
		System.out.println(name + " " + Surname + " aged "+age+".");

		if(!(dukeOf==null)){
		System.out.print("Is duke of "+dukeOf+". ");
		}
		if(! (viscountOf==null)){
			System.out.print("Viscount de "+viscountOf+". ");
		}
		if(!(lordName ==null)){
			System.out.print("Is known as " + lordName);
			if(!(lordOf==null))
				System.out.print(" lord of " + lordOf+". ");
			else
				System.out.print(". ");
		}
		if(!(knownFor==null)){
			System.out.print("Known for "+(gender?"His ":"her ")+"Deeds in the area of
"+knownFor);
		}

	}

	public static void main(String[] args){

		MedievalBuilder b = new MedievalBuilder();
		b.setAge(50);
		b.setName("Elias");
		b.setSurname("Thompson");


		b.setLordName(" Rayleigh");
		b.setLordOf("Staffordshire");
		b.setKnownFor("Physics");
		b.setGender(true);

		MedievalBuilderExample rayleigh = new MedievalBuilderExample(b);
		rayleigh.prettyPrint();
```

```
	}
}

//
MedievalBuilder.java
//

package uk.ac.cam.ap886.oopjava.supervision3;

public class MedievalBuilder{
	//Person - specific
	int age=0;
	String name=null;
	String Surname=null;

	//Optional parameters
	String dukeOf=null;
	String viscountOf=null;
	String lordName=null;
	String lordOf=null;
	boolean gender=false;
	String knownFor=null;


	public MedievalBuilder(){}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getSurname() {
		return Surname;
	}

	public void setSurname(String surname) {
		Surname = surname;
	}

	public String getDukeOf() {
```

```
		return dukeOf;
	}

	public void setDukeOf(String dukeOf) {
		this.dukeOf = dukeOf;
	}

	public String getViscountOf() {
		return viscountOf;
	}

	public void setViscountOf(String viscountOf) {
		this.viscountOf = viscountOf;
	}

	public String getLordName() {
		return lordName;
	}

	public void setLordName(String lordName) {
		this.lordName = lordName;
	}

	public String getLordOf() {
		return lordOf;
	}

	public void setLordOf(String lordOf) {
		this.lordOf = lordOf;
	}

	public boolean isGender() {
		return gender;
	}

	public void setGender(boolean gender) {
		this.gender = gender;
	}

	public String getKnownFor() {
		return knownFor;
	}

	public void setKnownFor(String knownFor) {
		this.knownFor = knownFor;
	}



}
```

----------**Question 9**

A common anti patterns that afflicts many novice programmers coming form languages such as C++ is the Blob. the above example contains two classes of both of which, perform tasks of relatively the same level (considering the second one is actually a container that encapsulates data. However most programmers tend to create one huge instance of of code that contains almost everything the project can do, and keeping clasees that perform minute tasks, and could hypothertically be merged with the main "Blob".

This is one of the main reasons why Java enforces strictly taht a project has to contain packages, packages can contain classes and each class HAS to be a separate file. C~ took this even further providing different extensions for classes with different purpose. e.g. classes that actually merely describe the interface of a program are called interfaces, as opposed to instantiable classes.

ANother ANti pattern is the Telescoping constructor patten that is counteracted by the builder class' presence.

In software engineering another famous design anti-pattern is the presence of "poltergeists" e.g. short lived classes that perform minute tasks such as opening and closing resources. THis causes lots of confustion as they are present and most classes are dependent on them, however in effect all tey do is occupy space on the heap. The most reliabel solution is redistribution of responsibilities between different clases.

----------**Question 11**

The followig project is a cyberPet game. The interface appears to be serial, in terms that it has several prompt-called functions. In reality these functions can very well be replaced by simply reading a parallel implementation gui interface (without actually having to prompt the user in any way). The current implementation is command-driven in terms that instead of asking the user to press a particular key this works like a sort of language parser that can understand commands (e.g. play 0 will display the play information with animal 0) it also allocates pets into a hashmap so that invoking (play petname will be identical (actually much faster because this will save time in looking up the key from the name array.

The Following is a very Long listing of the Code:

```
//
UserInterface
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public interface UserInterface {
	public String	getPlayerName();
	public Pet[]	getPlayerPets();
	public Action	actionPrompt();
	public boolean		continuePrompt();
	public Meal	feedPrompt();
	//Done
}

//
```

Game
//
```
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

import java.util.Iterator;


public class Game {

	public static final int  SUPPORTED_PETS=3;
	private Player p;	//Could make an array in the future
	private UserInterface ui;

	public UserInterface getUi() {
		return ui;
	}

	public void setUi(UserInterface ui) {
		this.ui = ui;
	}

	public Game(){
		ui = new CommandLineInterface(this);
		Pet[] pets=	ui.getPlayerPets();
		p = new Player(ui.getPlayerName(),pets,this);
	}

	public Player getPlayer(){
		return p;
	}

	public int getSupportedPets() {
		return SUPPORTED_PETS;
	}

	public static void main(String[] args){
		Game g = new Game();
		//calls the constructor and creates a player. Plus SOme additional Maintenance
		//The class is structured in such a way taht you could create an array of games
		//and run all of them on different threads, all witgh their own player.
		//IF you want to there could be multiple players in a single game, in that case
		//there's no intersection between Different players' pets.
		do{
			try{
				g.getPlayer().act(g.getUi().actionPrompt());
			}catch(ArrayIndexOutOfBoundsException error){
				System.out.println("AN error occurred. retrying.");
				continue;
			}
			Iterator<Pet> it = g.getPlayer().getPets().values().iterator();
			while(it.hasNext()){
				Pet p = (Pet)it;
```

```
				p.update();
			}


		}while(g.getUi().continuePrompt());

	}

}

//
CyberPet
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public interface CyberPet {
	public void feed(Meal m);
	public void sleep();
	public void respond();
	public void play();
	public void update();
	//DONE
}

//
Dog
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class Dog extends Pet {

	public Dog(String newName,Game newGame){
		super(newName,newGame);
	}

	@Override
	public void play() {
		if(fatigue<=70){
			System.out.println("My FLuffy Friend "+ this.NAME+" Wants to play?");
			try{
				Thread.sleep(2000);
			}catch (InterruptedException e) {
				e.printStackTrace();
			}finally{
				System.out.println("
					");
				System.out.println("                    _
	");
				System.out.println("                    ,:'/   _..._                    ");
				System.out.println("                   // ( `~~-.._.'                    ");
				System.out.println("                   \\| /    6\\___          ");
				System.out.println("                   |     6      4        ");
```

```
                        System.out.println("                    |           /         ");
                        System.out.println("                     \\_       .--'        ");
                        System.out.println("                     (_'---'`)           ");
                        System.out.println("                     / `'---`()          ");
                        System.out.println("                   ,'        |           ");
                        System.out.println("       ,          .'`        |           ");
                        System.out.println("       )\\       _.-'          ;          ");
                        System.out.println("      / |    .'`   _          /           ");
                        System.out.println("    /` /   .'      '.        , |          ");
                        System.out.println("   /  /   /          \\   ;   | |         ");
                        System.out.println("   | \\  |            | .|   | |         ");
                        System.out.println("    \\\\  `~|          /.-' |   | |         ");
                        System.out.println("     '-..-\\\\       _.;..._ |   |.;-.         ");
                        System.out.println("           \\\\    <`.._  )) |  .;-. ))       ");
                        System.out.println("           (__.  `  ))-'  \\\\_    ))'       ");
                        System.out.println("              `'--@`  jgs  `~~~          ");
                        System.out.println("                                         ");
                        System.out.println("                                         ");
                        System.out.println("                                         ");
                        System.out.println("                                         ");
                        System.out.println("                                         ");
                        System.out.println("                                         ");
                }
                this.fatigue+=23;
                //Dogs are generally eager pets;
        }
        else{
                System.out.println("I'm too tired to play... Woof!");
        }
}

@Override
public void sleep() {
        System.out.print("Even The Eager souls of Dogs sometimes get tired");
        System.out.print(".");
        this.fatigue= 0;
        try{
                Thread.sleep(500);
                System.out.print(".");
                this.satiation+=10;//So that a dog doesn't die in its sleep
                Thread.sleep(500);
                System.out.println(".");
        } catch (InterruptedException e) {
                e.printStackTrace();
        }

}

@Override
public String toString(){
        return "Dog, named " +NAME;
}
```

```
}

//
Fish
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class Fish extends Pet {

	public Fish(String newName,Game newGame){
		super(newName,newGame);
	}

	@Override
	public void play() {
		if(fatigue<=70){
			System.out.println("You sure are fun to watch, "+ this.NAME+".");
			try{
				Thread.sleep(2000);
			}catch (InterruptedException e) {
				e.printStackTrace();
			}finally{
				System.out.println("     .--~~```                ```~~--.                ");
				System.out.println("          ':--..___            ___..--:'           ");
				System.out.println("             \\      ```~~~~~~~```      /                 ");
				System.out.println("           .-`  ___.....-----.....___  '-.                ");
				System.out.println("         .:-~~``     ~          ~    ``~~-:.         ");
				System.out.println("        /`-..____ ~        ~         ~____..-'\\       ");
				System.out.println("       /  ~    '`-----.........---\"\"`        \\       ");
				System.out.println("      ;                                      ;        ");
				System.out.println("     ; '::.   '          _,          _,       ;       ");
				System.out.println("     |   ':::    '     .' (    ~   .-'./    ~ |       ");
				System.out.println("     |~  .:'   .     _/..._'.    .'.-'/        |       ");
				System.out.println("     | .:'       .-'`      ` '-./.'_.'         |       ");
				System.out.println("     |  ':.     ( o)   ))      ;= <_           |       ");
				System.out.println("     ; '::.      '-.,\\\\__ __.-;`\'. '.  .      ;       ");
				System.out.println("      ;    ':         \\) |`\\ \\)  '.'-.\\        ;       ");
				System.out.println("       \\.:'.:':.         \\_/       '-._\\     /        ");
				System.out.println("        \\ ':.     ~                   `    /        ");
				System.out.println("         '. '::..  _ . - - -- .~ _       ~ .'        ");
				System.out.println("           '-._':'                `'-_.-'        ");
				System.out.println("       jgs    (``\"--..._____...--\"``)        ");
				System.out.println("               `\\\"--...__    __...--\\\"`        ");
				System.out.println("                     `````                          ");
				System.out.println("
					");

			}
			this.fatigue+=11;
			//Fish don't play you just observe them
		}
		else{
```

```
				System.out.println("Gzrp... I need some sleep!");
			}

		}

		@Override
		public void sleep() {
			System.out.print("Surprisingly fish sleep too");
			System.out.print(".");
			this.fatigue= 0;
			try{
				Thread.sleep(500);
				System.out.print(".");
				//this.satiation+=10;
				//A fish on the other hand can very well die in its sleep
				Thread.sleep(500);
				System.out.println(".");
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

		}

		public String toString(){
			return NAME + " the Fish";
		}
	}


	//
	KungFuPanda
	//
	package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

	public class KungFuPanda extends Pet {

		public KungFuPanda(String newName,Game newGame){
			super(newName,newGame);
		}

		@Override
		public void play() {
			if(fatigue<=20){
				System.out.println("You sure are fun to watch, "+ this.NAME+".");
				try{
					Thread.sleep(2000);
				}catch (InterruptedException e) {
					e.printStackTrace();
				}finally{


		System.out.println("░░░░░░░░█░▀▀▀▄█▄░░░░░░░░░░░░░ ");
```

```
System.out.println("██▄░░░░░█░░░░░░▀▀▀▄░░░░░░░░░░░░░ ");
System.out.println("░█▀▄▄░░█▀▄░░▄░░░░░█░░░░░░░░▄▄▄██ ");
System.out.println("░█▄▄▄▀█▀█▀░▀██░░░█▄█▄░░░▄█████▀▀ ");
System.out.println("▀█▄████▄░▄▄░░░░▄▄████████████▀░░ ");
System.out.println("░▀██████▄▄▄▄▄██████████████▄█░░░ ");
System.out.println("░░░▀█████████████████████▀▀░░░░░ ");
System.out.println("░░░░░▀▀██████████████▀▀░░░░░░░░░ ");
System.out.println("░░░░░░░█▀▀▀▀▀░░▀░░░▀█░░░░░░░░░░░ ");
System.out.println("░░░░░░█░░░░░░░░░░░░░░█░░░░░░░░░░ ");
System.out.println("░░░░░█░░░░░░░░░░░░░░░█░░░░░░░░░░ ");
System.out.println("░░░░░█░░░░░░░░░░░░░░░█░░░░░░░░░░ ");
System.out.println("░░░▄████▄▄░░░░░░░░░░██░░░░░░░░░░ ");
System.out.println("░░▄████████░░░░░░░▄███░░░░░░░░░░ ");
System.out.println("░░█████████▀▀▀▀███████░░░░░░░░░░ ");
          System.out.println("░░███████░░░░░░▀███████░░░░░░░░░ ");
          System.out.println("░░▀▀█████░░░░░░░▀▀████▀░░░░░░░░░ ");
          //Kung fu Panda, sorry, no Ninja warrior on the net
            }
            this.fatigue+=80;
      }   //A KungFuPanda really has to wear itself out
      else{
            System.out.format("What. No no more bamboo mommy! %n"
                        + "I don't want to go to school%n");
      }



}

@Override
public void sleep() {
      System.out.print("Pandas need more sleep");
      System.out.print(".");
      this.fatigue= 0;
      try{
            Thread.sleep(500);
            System.out.print(".");
            //this.satiation+=10;
```

```
                        //A fish on the other hand can very well die in its sleep
                        Thread.sleep(500);
                        System.out.print(".");
                        for(int i=0;i<4;i++){
                                Thread.sleep(500);System.out.print(".");
                        }
                        System.out.println();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }

        }

        public String toString(){
                return "Master "+NAME;
        }

}

//
Pet
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public abstract class Pet implements CyberPet{

        protected Game game;
        protected final String NAME;
        protected int fatigue = 30;
        protected int satiation = 100;

        //----------------------------------------



        public Pet (String newName,Game newGame){
                NAME = newName;
                game = newGame;
        }

        @Override
        public void feed(Meal m) {
                if(satiation >100){
                        System.out.println("The Poor "+ this+" seems to be full");
                }else if(satiation>60){
                        System.out.println(this + " happily eats the "+ m.toString().toLowerCase());
                }else if(satiation>0){
                        System.out.println(NAME + " seems to be extremely hungry.");
                }else{
                        System.out.println("It's no use feeding a dead pet.");
                }
```

```
	}

	@Override
	public abstract void play();

	@Override
	public void respond(){
		String tired =fatigue>50? " tired and sleepy":" Ready to play";
		String conjunctor;
		String hungry;
		if(satiation >100){
			hungry = " overfed";
			conjunctor = fatigue>50?" as well as":" but";
		}else if(satiation>60){
			hungry = " well nourished";
			conjunctor = fatigue>50?" though":" and";
		}else if(satiation>0){
			hungry = " hungry";
			conjunctor = fatigue>50?" as well":" and definitely not";
		}else{
			System.out.println("A dead pet cannot speak.");
			return;
		}
		System.out.println("Hi, "+game.getPlayer().getName()+"! I'm
currently"+hungry+conjunctor+tired);
	}

	@Override
	public abstract void sleep();

	public void update(){
		satiation-=3;
		fatigue+=4;
	}

	//DONE

	}


//
Player
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

import java.util.HashMap;

public class Player {
	private String name;
	private Game game;
	private String[] petNames;
	private HashMap<String,Pet> pets=new HashMap<String,Pet>();
```

```
	//At this point support only three pets.

	//-------------------------------------------------
	public Player (String newName, Pet[] newPets, Game newGame){
		//Want to avoid passing the reference.
		game = newGame;
		petNames = new String[Game.SUPPORTED_PETS];
		setName(newName);
		try{
		for(int i=0;i<newPets.length;i++){
			petNames[i]=new String(newPets[i].NAME);
			pets.put(newPets[i].NAME, newPets[i]);
		}
		}catch (IndexOutOfBoundsException error){
			System.out.println("Internal error::");
		}

	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public void act (Action predicate){
		UserInterface ui = game.getUi();
		String petName = predicate.PET_NAME==null?
petNames[predicate.PET_INDEX]:predicate.PET_NAME;
		if(!pets.containsKey(petName )){
			throw new IndexOutOfBoundsException();
		}
		else{
		switch (predicate.ACTION_TYPE.toLowerCase().trim()){
		case "play":
			pets.get(petName).play();
			break;
		case "feed":
			pets.get(petName).feed(ui.feedPrompt());
			break;
		case "sleep":
		case "let be":
			pets.get(petName).sleep();
			break;
		case "status":
		case "check":
			pets.get(petName).respond();
			break;
		default:
				System.out.println("Unrecognised action. Defaulting to update");
```

```
                }
                }

        }

        public HashMap<String,Pet> getPets() {
                return pets;
        }


        //Done
}



//
Action
//

package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class Action {
        /*
         * Not a true class, more of a strucutre. contains PET_INDEX;  ACTION_TYPE
         * */

        public final int          PET_INDEX;
        public final String ACTION_TYPE;
        public final String PET_NAME;
        public Action(int newIndex, String newType){
                PET_INDEX = newIndex;
                ACTION_TYPE= newType;
                PET_NAME = null;
        }

        public Action(String newPetName,String newType){
                PET_INDEX = -1;
                ACTION_TYPE= newType;
                PET_NAME = newPetName;
        }
        //DONE
}

//
CommandLineInterface
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
public class CommandLineInterface implements UserInterface {

	private BufferedReader c = new BufferedReader(new InputStreamReader(System.in));
	private Game game;

	//-------------------------------------------------------------------------

	public CommandLineInterface (Game newGame){
		game = newGame;
	}

	@Override
	public String getPlayerName() {
		System.out.format("%n------------------%nPlease Type in Your Name %n -> ");
		try {
			return  c.readLine();
		} catch (IOException e) {
			e.printStackTrace();
			return null;
		}
	}

	@Override
	public Pet[] getPlayerPets() {
		Pet[] pets = new Pet[Game.SUPPORTED_PETS];
		String petType=new String();
		String petName=new String();
		for(int i=0;i<Game.SUPPORTED_PETS;i++){
			System.out.print("Please Choose type of pet_"+i+" -> "); //Would look better
in Latex
			try {
				petType = c.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
			System.out.print("How would you like to name your pet -> ");
			try {
				petName = c.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
			pets[i] = parseToPet(petType, petName);

			//DEBUG
			System.out.println("Created "+ pets[i]);
		}
		return pets;
	}

	private Pet parseToPet(String petType,String petName) {
		switch (petType.toLowerCase().trim()){
```

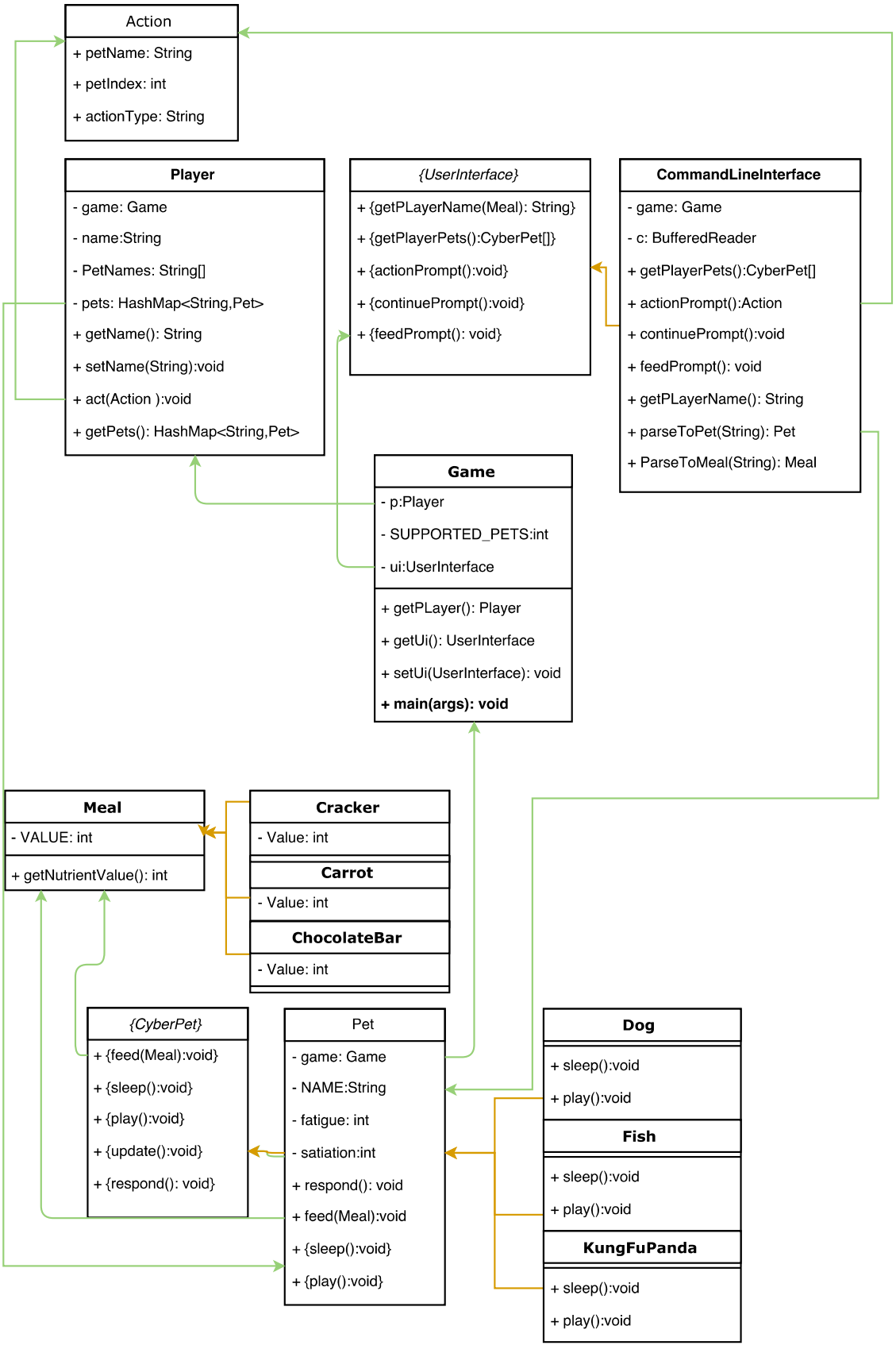
```
            case "kung fu":
            case "kungfu":
            case "ninja":
            case "panda":
                return new KungFuPanda(petName,game);
            case "fish":
            case "fishie":
            case "fisher":
            case "aquarium":
                return new Fish(petName,game);
            case "dog":
            case "doge":
            case "doggie":
            default:
                return new Dog(petName,game);
            }
    }

    private Meal parseToMeal(String mealName){
            switch (mealName.toLowerCase()){
            case "snickers":
            case "mars":
            case "bounty":
            case "treat":
            case "chocolate":
                return new ChocolateBar();
            case "carrot":
            case "veg":
            case "vegetable":
                return new Carrot();
            case "cracker":
            case "food":
            case "default":
            default:
                return new Cracker();
            }
    }

    @Override
    public Action actionPrompt() {
            System.out.format("Please Specify what you want to do:%n"
                            + "[Action Keyword] [Pet Number]%n%n");
            String[] input;
            try {
                input = c.readLine().split(" ");
            } catch (IOException e) {
                e.printStackTrace();
                return null;
            }
            int petIndex = 0;//Defaults to 0;
            while(input.length<2){
```

```
                System.out.println("Invalid input. Should be [Action Keyword] [Pet
Number]");
                try {
                        input = c.readLine().split(" ");
                } catch (IOException e) {
                        e.printStackTrace();
                        return null;
                }
        }
        try{
                petIndex = Integer.parseInt(input[1]);
                if(petIndex<0){
                        throw new NumberFormatException();
                }
        }catch(NumberFormatException error){
                return new Action(input[1],input[0]);
        }

        //Action String Recognition should be in the Act Method


        return new Action(petIndex, input[0]) ;
}

@Override
public boolean continuePrompt() {
        System.out.println("Would you like to quit?");
        String input=new String();
        try {
                input = c.readLine();
        } catch (IOException e) {
                e.printStackTrace();
        }
        switch (input.toLowerCase()){
        case "yes":
        case "oui":
        case "yup":
        case "y":
        case "yeah I'd like to quit, please":
                return false;
        default:
                return true;
        }
}

@Override
public Meal feedPrompt(){
        //In future would allow to select what to feed. now feeds
        //the last thing on the list.
        System.out.println("What would you like to feed?");
        try {
                return parseToMeal(c.readLine());
```

```
                } catch (IOException e) {
                        e.printStackTrace();
                        return null;
                }


        }


}
//
Meal
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public abstract class Meal  {
        private int VALUE;    //Intrinsically immutable therefore constant

        protected Meal(int newValue){
                VALUE = newValue; //Shouldn't be able to create uncreative meals
                                                        //Don't have default appearance, name therefore
        }
        //------------------------
        public int getNutrientValue(){
                return VALUE;
        }
        //DONE
}

//
Carrot
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class Carrot extends Meal {
        public Carrot(){
                super (20);
        }

        public String toString(){
                return "Carrot";
        }
        //DONE
}

//
ChocolateBar
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class ChocolateBar extends Meal {
```

```
	public ChocolateBar(){
		super (100);
	}

	@Override
	public String toString(){
		return "Chocolate Bar";
	}
	//DONE
}

//
Cracker
//
package uk.ac.cam.ap886.oopjava.supervision.cyberPet;

public class Cracker extends Meal {
	public Cracker (){
		super (5);
	}

	public String toString(){
		return "Cracker";
	}
	//DONE
}
```

----------Question 10


Action
+ petName: String
+ petIndex: int
+ actionType: String
Player
- game: Game
- name:String
- PetNames: String[]
- pets: HashMap<String,Pet>
+ getName(): String
+ setName(String):void
+ act(Action):void
+ getPets(): HashMap<String,Pet>
{UserInterface}
+ {getPLayerName(Meal): String}
+ {getPlayerPets():CyberPet[]}
+ {actionPrompt():void}
+ {continuePrompt():void}
+ {feedPrompt(): void}
CommandLineInterface
- game: Game
- c: BufferedReader
+ getPlayerPets():CyberPet[]
+ actionPrompt():Action
+ continuePrompt():void
+ feedPrompt(): void
+ getPLayerName(): String
+ parseToPet(String): Pet
+ ParseToMeal(String): Meal
Game
- p:Player
- SUPPORTED_PETS:int
- ui:UserInterface
+ getPLayer(): Player
+ getUi(): UserInterface
+ setUi(UserInterface): void
+ main(args): void
Meal
- VALUE: int
+ getNutrientValue(): int
Cracker
- Value: int
Carrot
- Value: int
ChocolateBar
- Value: int
{CyberPet}
+ {feed(Meal):void}
+ {sleep():void}
+ {play():void}
+ {update():void}
+ {respond(): void}
Pet
- game: Game
- NAME:String
- fatigue: int
- satiation:int
+ respond(): void
+ feed(Meal):void
+ {sleep():void}
+ {play():void}
Dog
+ sleep():void
+ play():void
Fish
+ sleep():void
+ play():void
KungFuPanda
+ sleep():void
+ play():void