Statement of Work:
We want to create a vampire hunter game and our character in this game hunts those vampires in different ways. To give an example, as the most popular option, he can kill those vampires by putting a stake in his heart. Killing them by excluding them from sunlight is one of the common methods used especially in movies. Another option is to kill by burning. An uncertain way is to use holy water. This should also be in the options. These options can be increased or changed. Perhaps different vampire types may be added in the future.

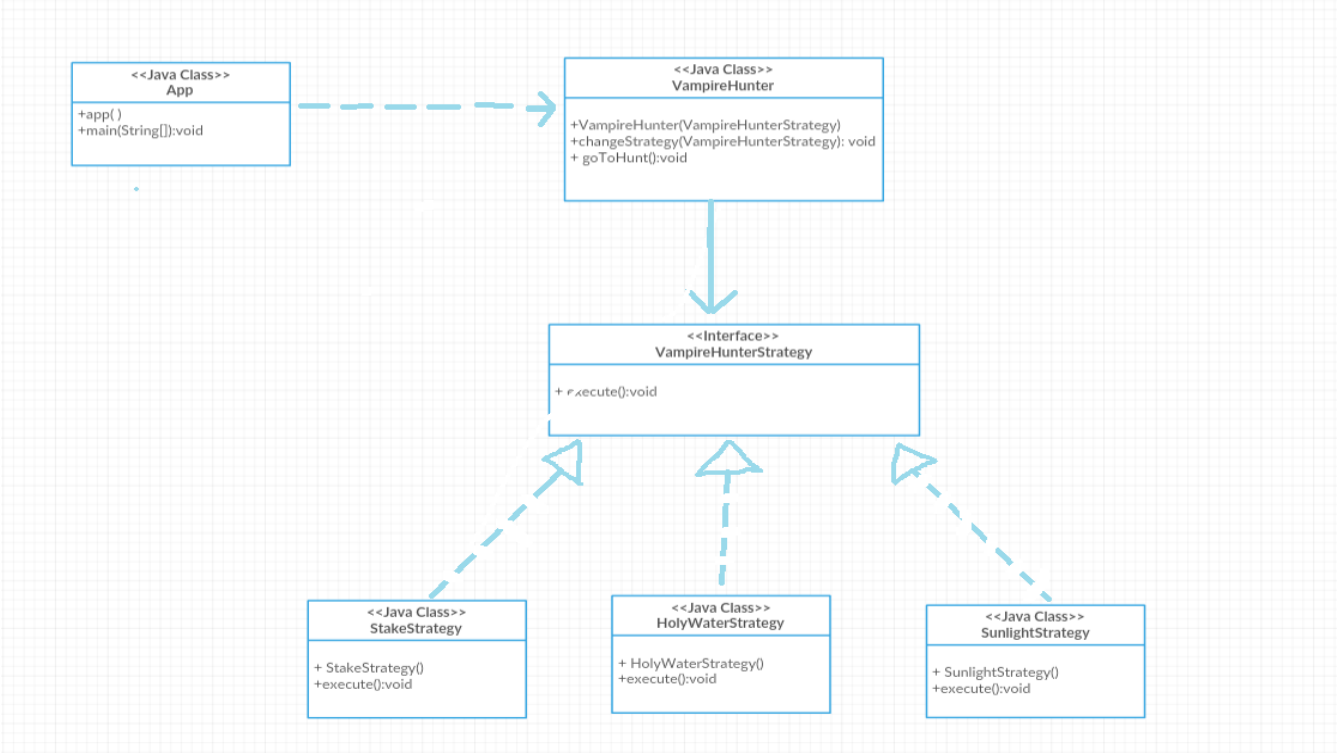
Design Pattern:

Strategy pattern

I used strategy pattern in this project because we have multiple algorithms used to kill vampires. like putting stake, using holywater, sunlight .. I created a VampireHunterStrategy interface and defined the common methods to be used in all classes in the interface. I then created algorithm classes to implement the interface created. StakeStrategy, holyWaterStrategy etc. Then I created the

VampireHunter class. Within this class, there is a method for setting algorithms of algorithm interface type. With this method, the VampireHunter class holds the algorithm object. Finally, I created an App class. This class generates objects from the VampireHunter class and algorithm classes, allowing operations to be performed.

3-UML



4-Research

➔ package com.journaldev.design.strategy;

```
public interface PaymentStrategy {

	public void pay(int amount);
}
```

➔ package com.journaldev.design.strategy;

```
public class CreditCardStrategy implements PaymentStrategy {

	private String name;
	private String cardNumber;
	private String cvv;
	private String dateOfExpiry;

	public CreditCardStrategy(String nm, String ccNum, String cvv, String expiryDate){
		this.name=nm;
		this.cardNumber=ccNum;
		this.cvv=cvv;
		this.dateOfExpiry=expiryDate;
	}
	@Override
	public void pay(int amount) {
		System.out.println(amount +" paid with credit/debit card");
	}

}
```

➔ package com.journaldev.design.strategy;

```
public class PaypalStrategy implements PaymentStrategy {

	private String emailId;

	private String password;
```

```
	public PaypalStrategy(String email, String pwd){
		this.emailId=email;
		this.password=pwd;
	}

	@Override
	public void pay(int amount) {
		System.out.println(amount + " paid using Paypal.");
	}

}
```

➔ package com.journaldev.design.strategy;

```
public class Item {

	private String upcCode;
	private int price;

	public Item(String upc, int cost){
		this.upcCode=upc;
		this.price=cost;
	}

	public String getUpcCode() {
		return upcCode;
	}

	public int getPrice() {
		return price;
	}
```

}

➔ package com.journaldev.design.strategy;

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public class ShoppingCart {

	//List of items
	List<Item> items;

	public ShoppingCart(){
		this.items=new ArrayList<Item>();
	}

	public void addItem(Item item){
		this.items.add(item);
	}

	public void removeItem(Item item){
		this.items.remove(item);
	}

	public int calculateTotal(){
		int sum = 0;
		for(Item item : items){
			sum += item.getPrice();
		}
		return sum;
```

```
	}

	public void pay(PaymentStrategy paymentMethod){
		int amount = calculateTotal();
		paymentMethod.pay(amount);
	}
}
```

➔ package com.journaldev.design.strategy;

```
public class ShoppingCartTest {

	public static void main(String[] args) {
		ShoppingCart cart = new ShoppingCart();

		Item item1 = new Item("1234",10);
		Item item2 = new Item("5678",40);

		cart.addItem(item1);
		cart.addItem(item2);

		//pay by paypal
		cart.pay(new PaypalStrategy("myemail@example.com", "mypwd"));
		//pay by credit card
		cart.pay(new CreditCardStrategy("Pankaj Kumar", "1234567890123456", "786",
"12/15"));
	}

}
```

I found a project showing how we can implement strategy design in a shopping card.

In this shopping card app, First of all they create the interface,in case to pay the amount passed as argument. Now they create concrete implementation of algorithms for payment using credit/debit card or through paypal. They can implement Shopping Cart and payment method will require input as Payment strategy. Notice that payment method of shopping cart requires payment algorithm as argument and doesn't store it anywhere as instance variable.
Finally a class was created for the project test.

The Strategy pattern suggests that you take a class that does something specific in a lot of different ways and extract all of these algorithms into separate classes called strategies.

This pattern allows that add new algorithms or modify existing ones without changing the code of the context or other strategies.