User Story:

As a Scrum team member, I want to refactor a larger switch statement that continually changes because of new conditions being added.

Motivation:

Using a switch statement sometimes implies spaghetti and very crowded code. The Strategy Pattern helps to divide an algorithm from a host class and then move it to another class.

**Before:**

public class Company

{

public void Checkout(IEnumerable<Product> products, Emlpoyee employee)

{

if (!employee.IsNotFlagged)

{

// the customer account is flagged

// log some errors and return

return;

}

if (!employee.IsNotPresent)

{

return;

}

}

}

public class Emlpoyee

{

public decimal Balance { get; private set; }

public bool Present { get; private set; }

public bool IsNotFlagged

{

get { return Balance < 30m; }

}

public bool IsNotPresent

{

get { return Present = true; }

}

}

public class Product

{

}

**Mechanics:**

The Strategy Pattern can also help us to replace a switch statement. Strategy Pattern can prevent the horror of using an endless switch or spaghetti code.

**After:**

public class Company

{

public void Checkout(IEnumerable<Product> products, Emlpoyee employee)

{

if (employee.IsFlagged)

{

return;

}

if (employee.IsPresent)

{

return;

}

}

}

public class Emlpoyee

{

public decimal Balance { get; private set; }

public bool Present { get; private set; }

public bool IsFlagged

{

get { return Balance < 30m; }

}

public bool IsPresent

{

get { return Present = true; }

}

}

public class Product

{

}