Template Design Pattern:-

When Algorithm of a program is fixed, but its implementation of function inside the algorithm varies (fill in the blanks type) then we call this pattern as a template design pattern.

```java
public abstract class TaxCalculator {
    public final int calculateTax(int income) { //final as all taxes are calculated in the same way.
        int incAfterSD = applyStandardDecuction(income);
        int tax = applyTaxRate(incAfterSD);
        int taxAfterSurcharge = applySurcharge(tax);
        return taxAfterSurcharge;
    }

    abstract int applyStandardDecuction(int income);
    abstract int applyTaxRate(int income);
    abstract int applySurcharge(int tax);
}
```

Here calculate tax algorithm is fixed but applyStandardDeduction, applyTaxRate and applySurcharge functions have different implementation according to different entities.

```java
public class YoungMaleTaxCalculator extends TaxCalculator {
    @Override
    int applyStandardDecuction(int income) {
        return income - 50000;
    }

    @Override
    int applySurcharge(int tax) {
        return (int)(tax * 1.02);
    }

    @Override
    int applyTaxRate(int income) {
        return (int)(income * 0.2);
    }
}
```

```java
public class YoungFemaleTaxCalculator extends TaxCalculator {
    @Override
    int applyStandardDecuction(int income) {
        return income - 30000;
    }

    @Override
    int applySurcharge(int tax) {
        return (int)(tax * 1.01);
    }

    @Override
    int applyTaxRate(int income) {
        return (int)(income * 0.2);
    }
}
```

```java
public class SeniorCitizenTaxCalculator extends TaxCalculator {
    @Override
    int applyStandardDecuction(int income) {
        return income - 100000;
    }

    @Override
    int applySurcharge(int tax) {
        return (int)(tax*1.0);
    }

    @Override
    int applyTaxRate(int income) {
        return (int)(income*0.1);
    }
}
```

different implementation entities derive from our base algorithm Tax Calculator class. and Hence we calculate Tax for different entities by passing their object to TaxCalculator base class.

```
class Test {
    Run | Debug
    public static void main(String[] args) {
        TaxCalculator tc1 = new YoungMaleTaxCalculator();
        TaxCalculator tc2 = new YoungFemaleTaxCalculator();
        TaxCalculator tc3 = new SeniorCitizenTaxCalculator();
        System.out.println("Young Male: " + tc1.calculateTax(income: 1000000));
        System.out.println("Young Female: " + tc2.calculateTax(income: 1000000));
        System.out.println("Senior Citizen: " + tc3.calculateTax(income: 1000000));
    }
}
```

```
Young Male: 193800
Young Female: 195940
Senior Citizen: 90000
```

Create a template for one algorithm. make fill in the blanks functions (abstract functions). And let entities define their implementation.

Proxy Design

Proxy is used for authentication or cache or web response proxy.
This is a class created exactly like the class we intend to proxy for and contains that class inside the proxy class as Composition.
We save the state/Bild of calls of functions inside the proxy class to make this class more smart for specific use cases.

```java
public interface ISomeWork {
    int fun1(int x);
}
```

```java
public class RealWork implements ISomeWork {
    @Override
    public int fun1(int x) {
        return x * x;
    }
}
```

```java
public class CacheProxyWork implements ISomeWork {
    RealWork rw = new RealWork();
    HashMap<Integer,Integer> f1map = new HashMap<>();

    @Override
    public int fun1(int x) {
        if(f1map.containsKey(x) == true){
            System.out.println(x: "Getting the already stored Data from Cache");
            f1map.get(x); //alse can return timestamp with this.
        }

        int res = rw.fun1(x);
        f1map.put(x,res);
        return res;
    }
```

```
class Test {
    Run | Debug
    public static void main(String[] args) {
        CacheProxyWork work = new CacheProxyWork();
        System.out.println(work.fun1(x: 5));
        System.out.println(x: "--------------------------");
        System.out.println(work.fun1(x: 5));
    }
}
```

```
25
--------------------------
Getting the already stored Data from Cache
25
```

Hence Proxy is just a class we use instead of the actual class so that we can add some more functionalities to Proxy class as use this class to our needs.