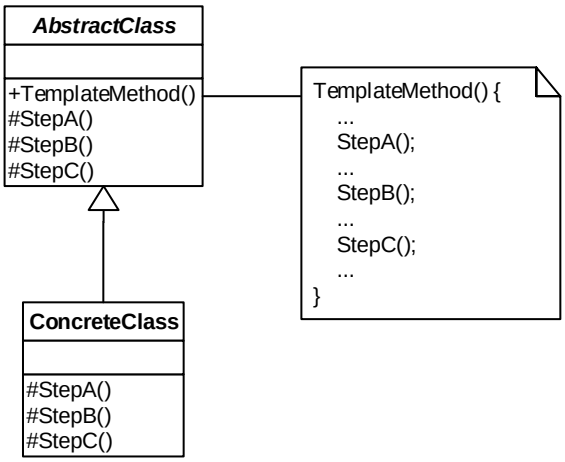
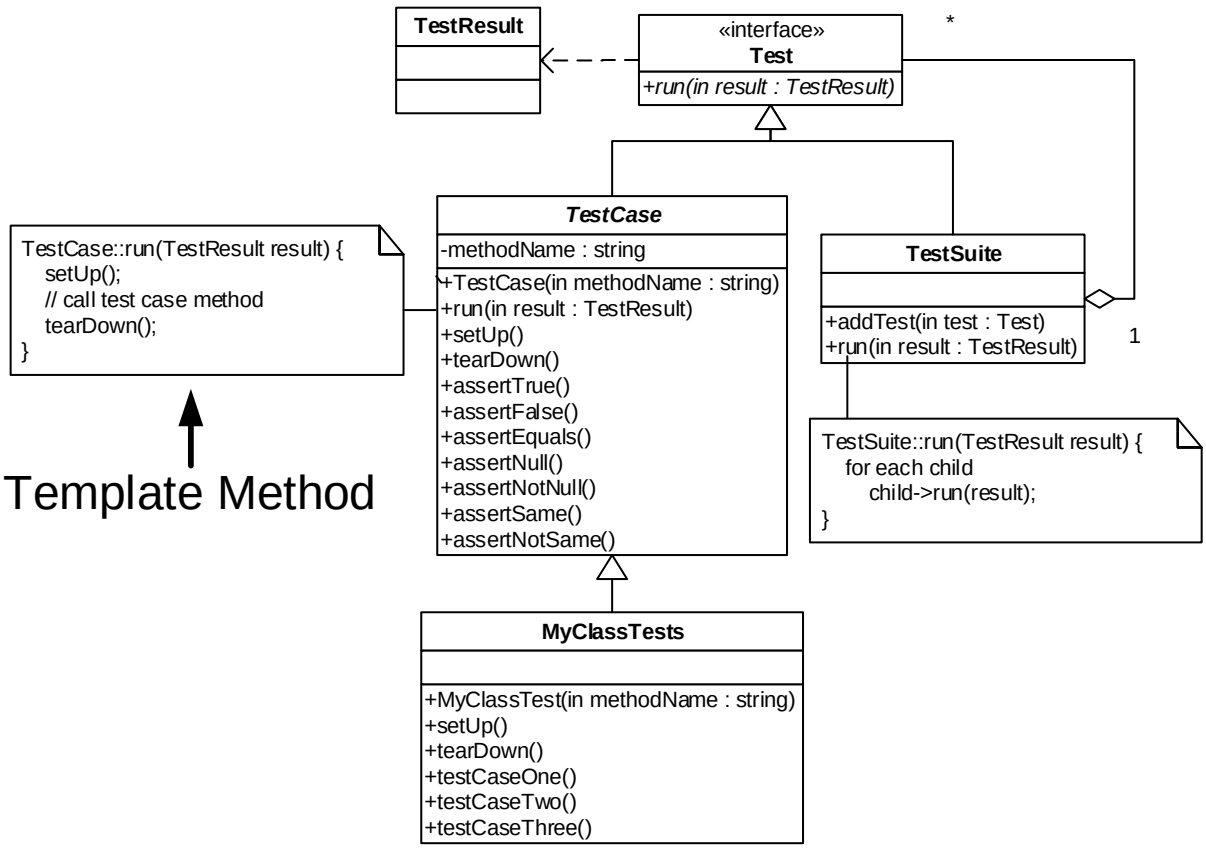
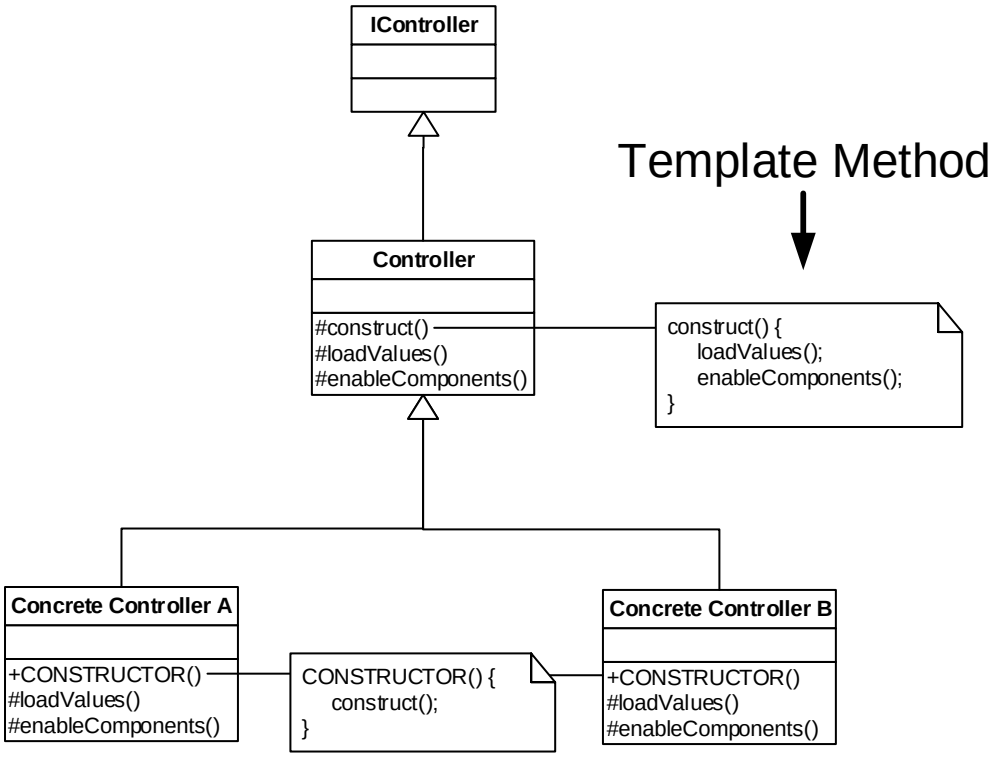
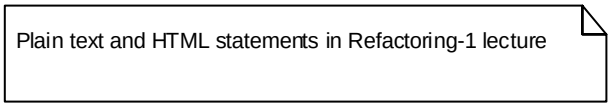

AbstractClass
+TemplateMethod()
#StepA()
#StepB()
#StepC()
TemplateMethod() {
...
StepA();
...
StepB();
...
StepC();
...
}
ConcreteClass
#StepA()
#StepB()
#StepC()


TestResult
«interface»
Test
+run(in result : TestResult)
*
TestCase
-methodName : string
+TestCase(in methodName : string)
+run(in result : TestResult)
+setUp()
+tearDown()
+assertTrue()
+assertFalse()
+assertEquals()
+assertNull()
+assertNotNull()
+assertSame()
+assertNotSame()
TestSuite
+addTest(in test : Test)
+run(in result : TestResult)
1
TestCase::run(TestResult result) {
setUp();
// call test case method
tearDown();
}
TestSuite::run(TestResult result) {
for each child
child->run(result);
}
Template Method
MyClassTests
+MyClassTest(in methodName : string)
+setUp()
+tearDown()
+testCaseOne()
+testCaseTwo()
+testCaseThree()


IController
Template Method
Controller
#construct()
#loadValues()
#enableComponents()
construct() {
loadValues();
enableComponents();
}
Concrete Controller A
+CONSTRUCTOR()
#loadValues()
#enableComponents()
CONSTRUCTOR() {
construct();
}
Concrete Controller B
+CONSTRUCTOR()
#loadValues()
#enableComponents()

```
/** * An abstract class that is common to several games in * which players play against the others, but only one is * playing at a given time. */ abstract class Game { private int playersCount; abstract void initializeGame(); abstract void makePlay(int player); abstract boolean endOfGame(); abstract void printWinner(); /* A template method : */ final void playOneGame(int playersCount) { this.playersCount = playersCount; initializeGame(); int j = 0; while (!endOfGame()){ makePlay(j); j = (j + 1) % playersCount; } printWinner(); } } //Now we can extend this class in order to implement actual games: class Monopoly extends Game { /* Implementation of necessary concrete methods */ void initializeGame() { // ... } void makePlay(int player) { // ... } boolean endOfGame() { // ... } void printWinner() { // ... } /* Specific declarations for the Monopoly game. */ // ... } class Chess extends Game { /* Implementation of necessary concrete methods */ void initializeGame() { // ... } void makePlay(int player) { // ... } boolean endOfGame() { // ... } void printWinner() { // ... } /* Specific declarations for the chess game. */ // ... }
```


Plain text and HTML statements in Refactoring-1 lecture