The Template Method Pattern

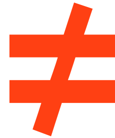
Human Computer Interaction Research
University of Nevada, Reno

Template method

- ★ Structural Patterns
  - » adapter
  - » façade
- ★ Behavioral Patterns
  - » strategy
  - » observer
  - » decorator
  - » command
  - » template method
- ★ Creational Patterns
  - » factory method
  - » abstract factory
  - » singleton

clarification

template method



```
template <typename T>
```

"advanced" C++ coding

Problem

duplicated code is difficult to change, maintain or extend

example

Class A

```
void do() {
   do_method_A();
   do_method_B();
   do_method_C();
   do_method_D();
}

void method_A() {
}
void method_B() {
}
void method_C() {
}
void method_D() {
}
```

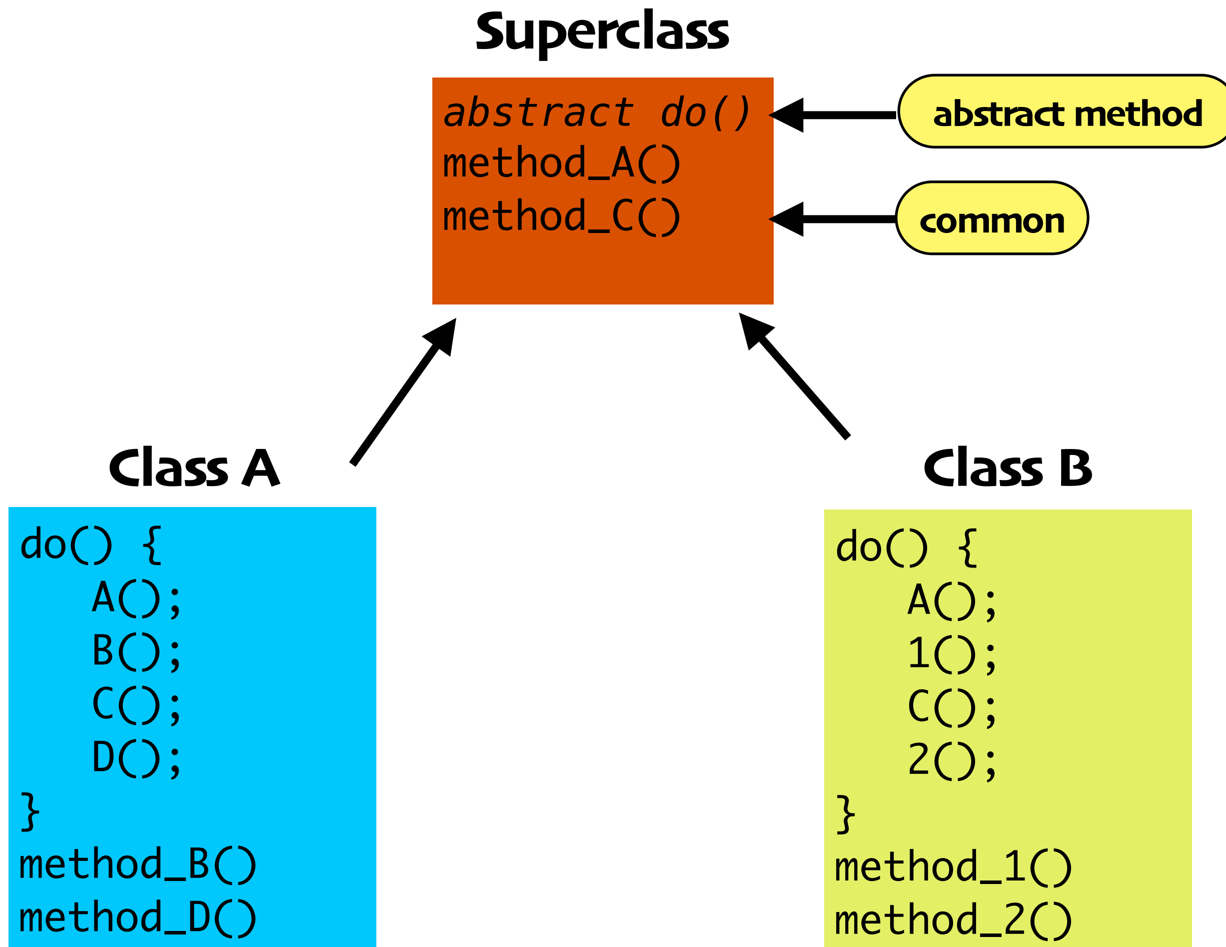
code duplication

Class B

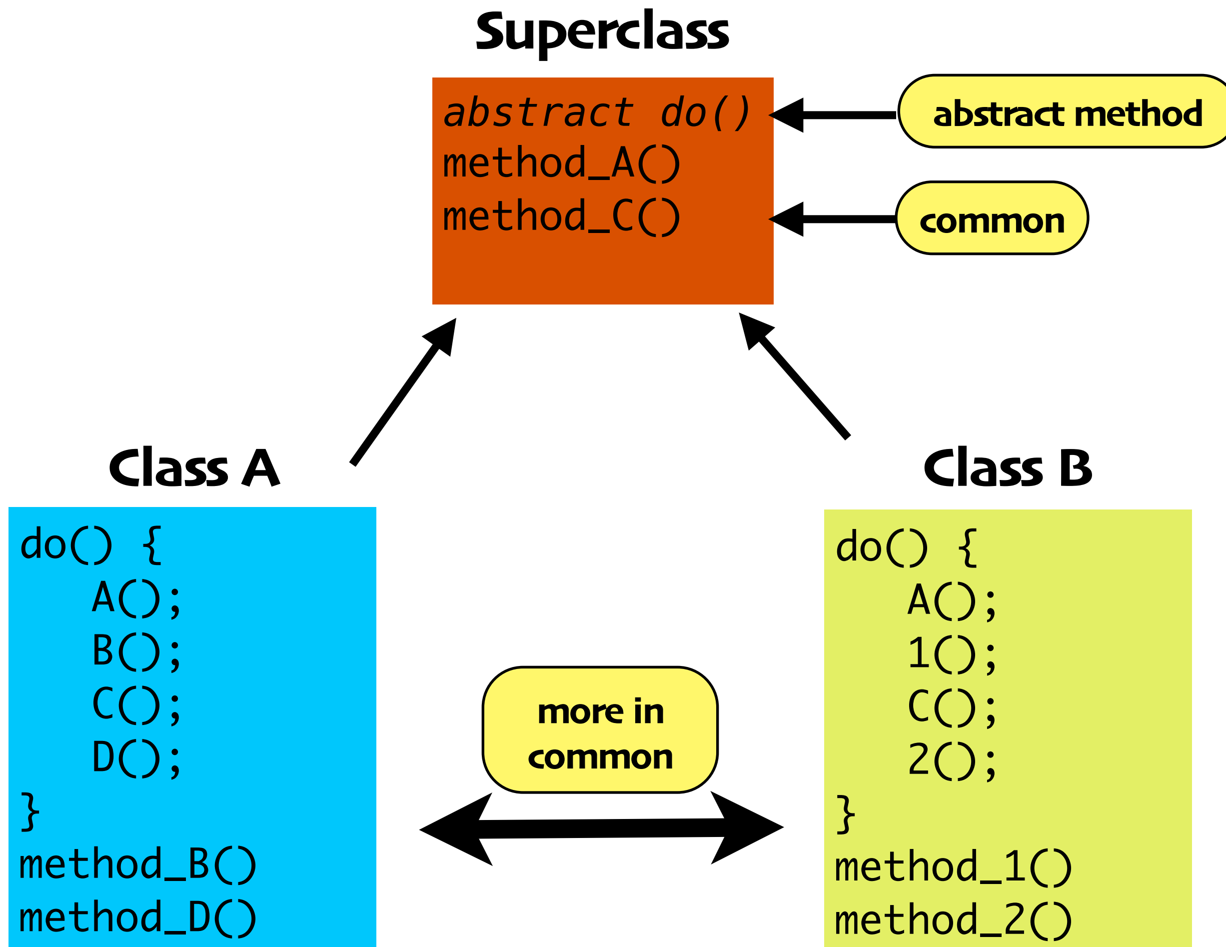
```
void do() {
   do_method_A();
   do_method_1();
   do_method_C();
   do_method_2();
}

void method_A() {
}
void method_1() {
}
void method_C() {
}
void method_2() {
}
```

Remove redundancy



Remove redundancy



Remove redundancy

Class A

```
do() {
   A();
   B();
   C();
   D();
}
method_B() {
  op1();
  op2();
  op3();
}
```

Class B

```
do() {
   A();
   1();
   C();
   2();
}
method_1() {
  op1();
  opA();
  op3();
}
```

same

code duplication

Define Template Method

Abstract Class

this is called a template method and its final to avoid subclasses overriding it

```
final do() {
   A();
   1B();
   C();
   2D();
}
abstract method_1B()
abstract method_2D()
method_A()
method_C()
```

must be implemented by concrete subclasses

primitive method

concrete method

Class A

```
method_1B()
method_2D()
```

override

Class B

```
method_1B()
method_2D()
```

Template Method

The Template Method Pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses.

Code

```
abstract class AbstractClass {
    /* A template method : */
    final void TemplateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
    }
    abstract void primitiveOperation1();
    abstract void primitiveOperation2();
    final void concreteOperation() {
      doSomething();
    }
}
```

generic

must be implemented by concrete subclasses

generic

Add a Hook

```
abstract class AbstractClass {
    /* A template method : */
    final void TemplateMethod() {
        primitiveOperation1();
        primitiveOperation2();
        concreteOperation();
        hook();
    }
    abstract void primitiveOperation1();
    abstract void primitiveOperation2();
    final void concreteOperation() {
      doSomething();
    }
    void hook() { }
}
```

must implement

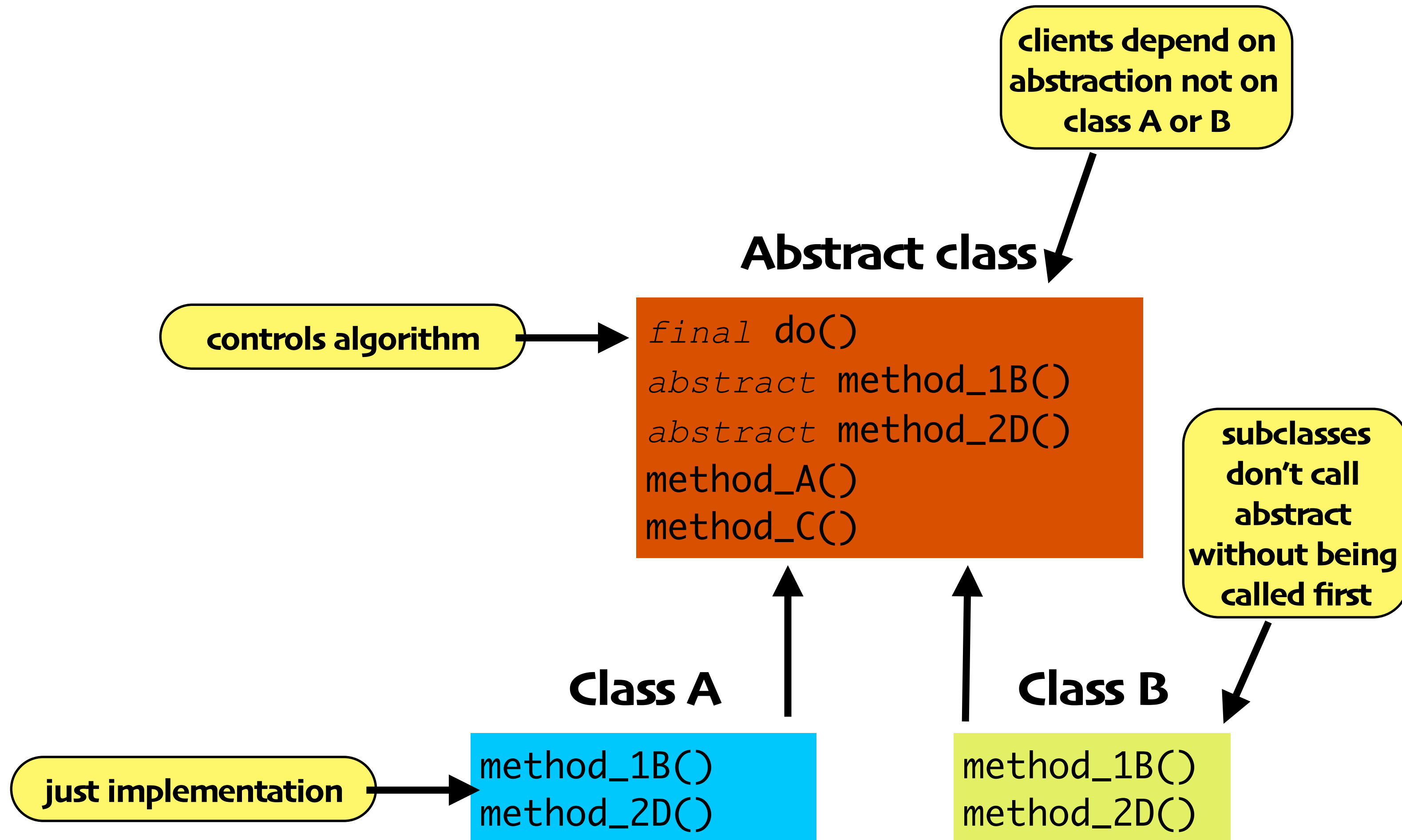
useful for logging or whatever

optional

stub

in the concrete class you could do a test to decide what the hook should do

Design Principle



Design Principle

The HollyWood principle: Don't call us, we'll call you

clients depend on abstraction not on class A or B

Abstract class

controls algorithm

```
final do()
abstract method_1B()
abstract method_2D()
method_A()
method_C()
```

subclasses don't call abstract without being called first

Class A

```
method_1B()
method_2D()
```

Class B

```
method_1B()
method_2D()
```

just implementation

Exercise 1

board games

abstract class game

```
abstract class Game {

    protected int playersCount;
    int winner;
    abstract void initializeGame();
    abstract void takeTurn(int player);
    abstract boolean endOfGame();
    abstract void collectResources();

    void declarewinner(){
       System.out.println("Player" + winner + "wins");
    }
```

template method

```
    /* A template method : */
    final void playOneGame(int playersCount) {
        this.playersCount = playersCount;
        initializeGame();
        int j = 0;
        while (!endOfGame()) {
            takeTurn(j);
            j = (j + 1) % playersCount;
        }
        declarewinner();
        collectResources();
    }
}
```

Concrete Game

```
public class Catan extends Game {
    static final int maxplayers = 5;
    int[] score = new int[maxplayers]; // inits to 0
    Random r = new Random();

    boolean endOfGame() {
        for (int i=0;i<maxplayers;i++) {
            if (score[i]==10) {
                winner=i;
                return true;
            }}
        return false;
    }

    void initializeGame() {
        System.out.println("initializing Catan");
    }
```

concrete game II

```
    void takeTurn(int player) {
        score[player]+=r.nextInt()%2;
        System.out.println("player+" + player + "takes a turn");
    }

    void collectResources() {
        System.out.println("collecting all sheep, clay, brick, wood and 
grain");
    }
}
```