Template Method Pattern

Prasanna Ghali

The Hot Beverage Problem

- Recipe for preparing coffee:
  - Boil water
  - Brew coffee grinds in boiling water
  - Pour coffee in cup
  - Add sugar and milk
- Recipe for preparing tea:
  - Boil water
  - Steep tea bag (or leaves) in boiling water
  - Pour tea in cup
  - Add honey and lemon

First Cut

```
class Coffee {
public:
  prepare_beverage();
private:
  boil_water();
  brew_coffee_grinds();
  pour_in_cup();
  add_sugar_and_milk();
};

void Coffee::prepare_beverage() {
  boil_water();
  brew_coffee_grinds();
  pour_in_cup();
  add_sugar_and_milk();
}
```

```
class Tea {
public:
  prepare_beverage();
private:
  boil_water();
  steep_tea_bag();
  pour_in_cup();
  add_honey_and_lemon();
};

void Tea::prepare_beverage() {
  boil_water();
  steep_tea_bag();
  pour_in_cup();
  add_honey_and_lemon();
}
```

What is Wrong?

```
class Coffee {
public:
  prepare_beverage();
private:
  boil_water();
  brew_coffee_grinds();
  pour_in_cup();
  add_sugar_and_milk();
};

void Coffee::prepare_beverage() {
  boil_water();
  brew_coffee_grinds();
  pour_in_cup();
  add_sugar_and_milk();
}
```

```
class Tea {
public:
  prepare_beverage();
private:
  boil_water();
  steep_tea_bag();
  pour_in_cup();
  add_honey_and_lemon();
};

void Tea::prepare_beverage() {
  boil_water();
  steep_tea_bag();
  pour_in_cup();
  add_honey_and_lemon();
}
```

What is Wrong?

```
class Coffee {
public:
  boil_water();
  brew_coffee_grinds();
  pour_in_cup();
  add_sugar_and_milk();
};


prepare_beverage(Coffee& c) {
  c.boil_water();
  c.brew_coffee_grinds();
  c.pour_in_cup();
  c.add_sugar_and_milk();
}
```

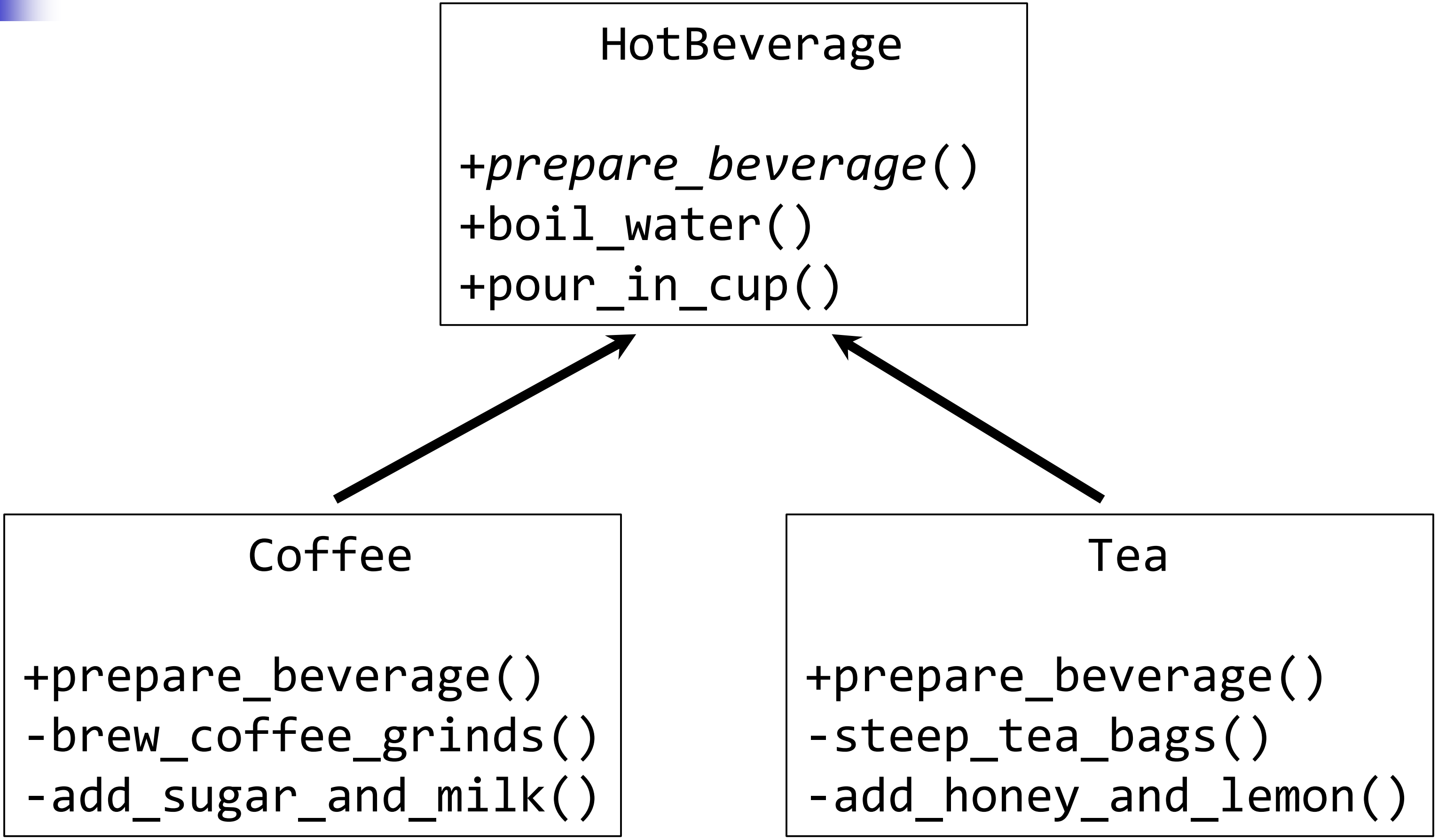
```
class Tea {
public:
  boil_water();
  steep_tea_bag();
  pour_in_cup();
  add_honey_and_lemon();
};


prepare_beverage(Tea& t) {
  t.boil_water();
  t.steep_tea_bag();
  t.pour_in_cup();
  t.add_honey_and_lemom();
}
```

Code Duplication

- `boil_water()` duplicated in both functions
- `pour_in_cup()` duplicated in both functions
- Code duplication implies imperfect design
- Commonality in both algorithms can be abstracted into a base class

First Cut at Redesign



HotBeverage

+*prepare_beverage*()
+boil_water()
+pour_in_cup()

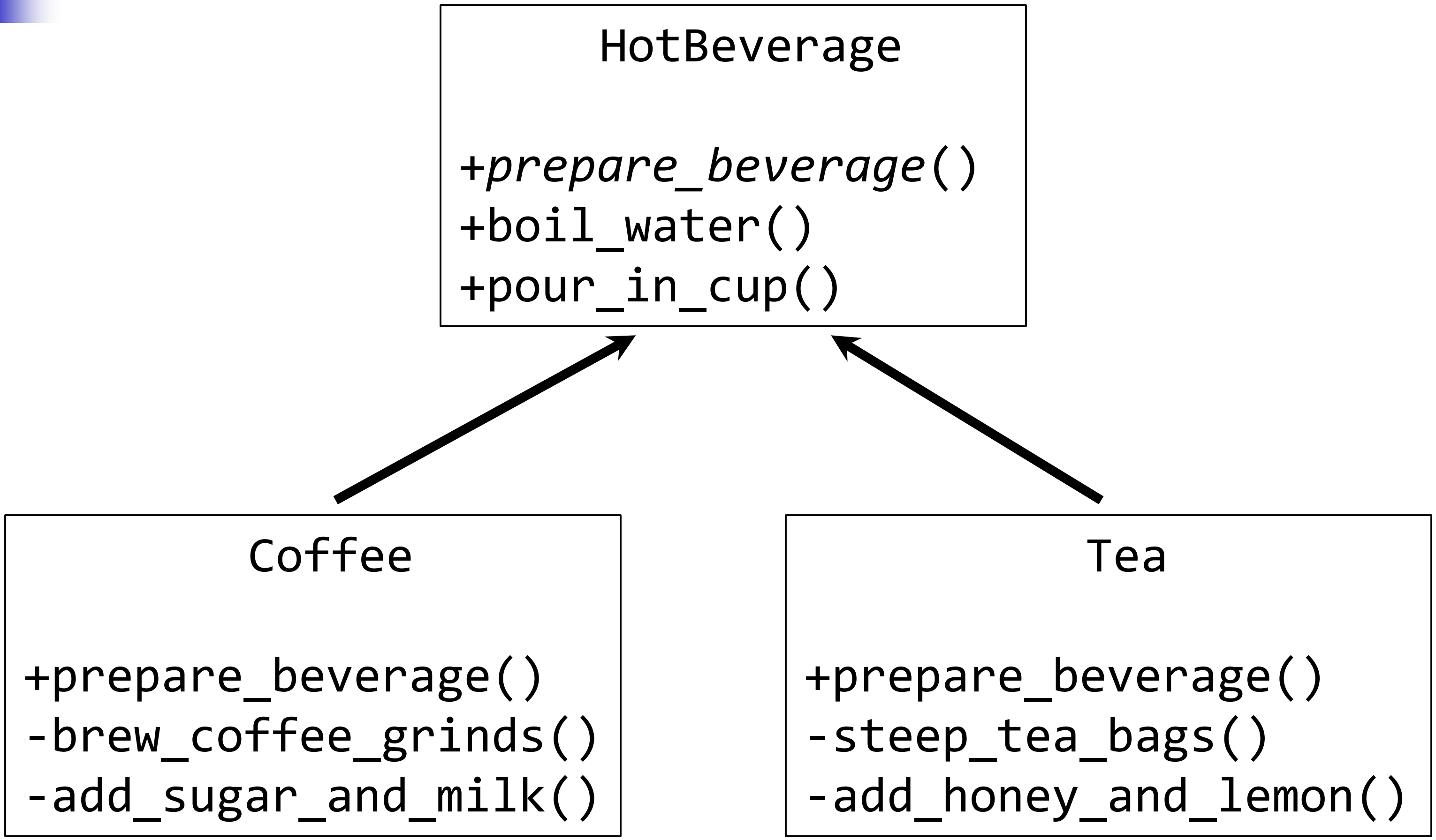
Coffee

+prepare_beverage()
-brew_coffee_grinds()
-add_sugar_and_milk()

Tea

+prepare_beverage()
-steep_tea_bags()
-add_honey_and_lemon()

Good Job On Redesign?



HotBeverage

+*prepare_beverage*()
+boil_water()
+pour_in_cup()

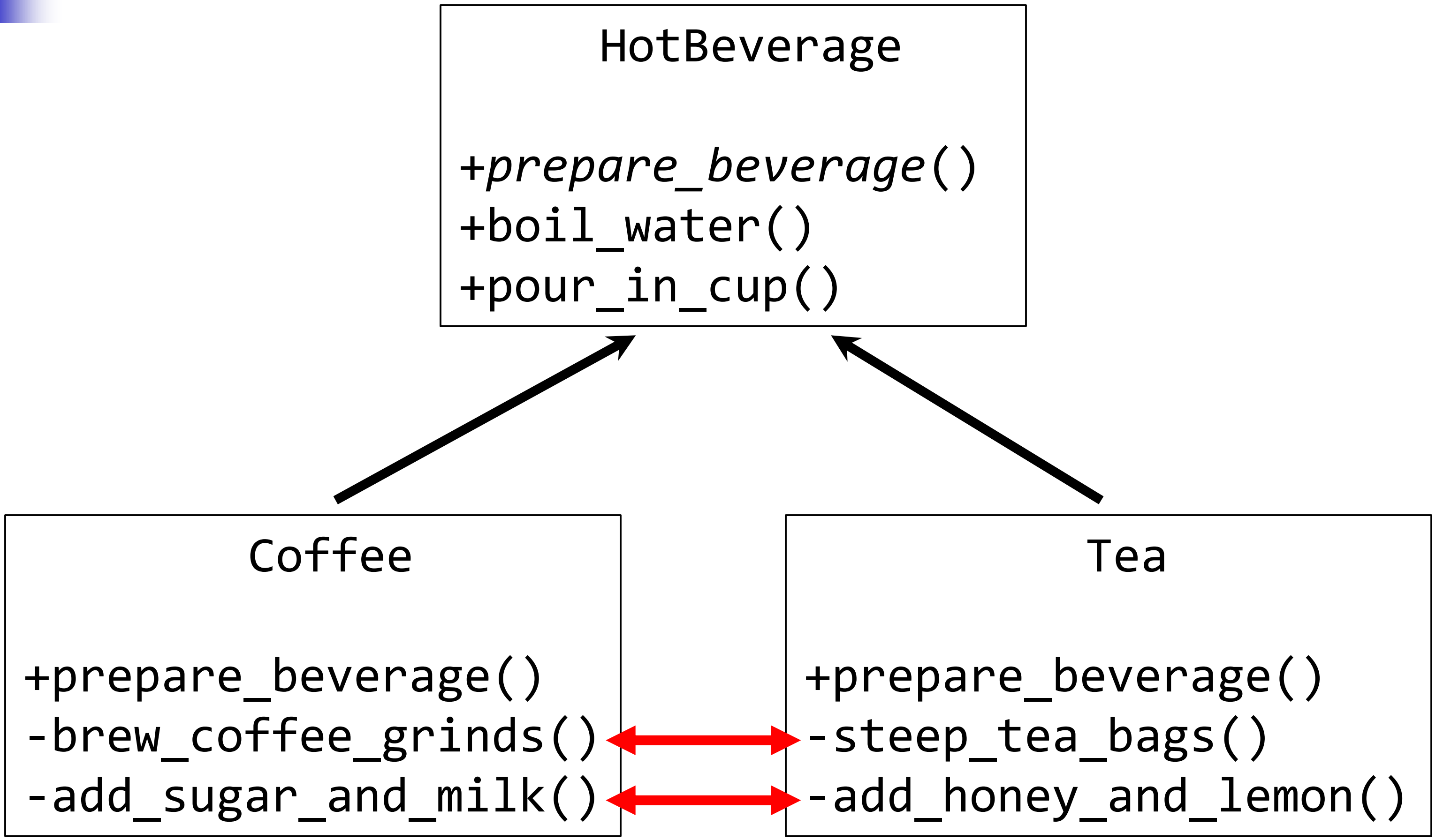
Coffee

+prepare_beverage()
-brew_coffee_grinds()
-add_sugar_and_milk()

Tea

+prepare_beverage()
-steep_tea_bags()
-add_honey_and_lemon()

Good Job On Redesign?



HotBeverage

+*prepare_beverage*()
+boil_water()
+pour_in_cup()

Coffee

+prepare_beverage()
-brew_coffee_grinds()
-add_sugar_and_milk()

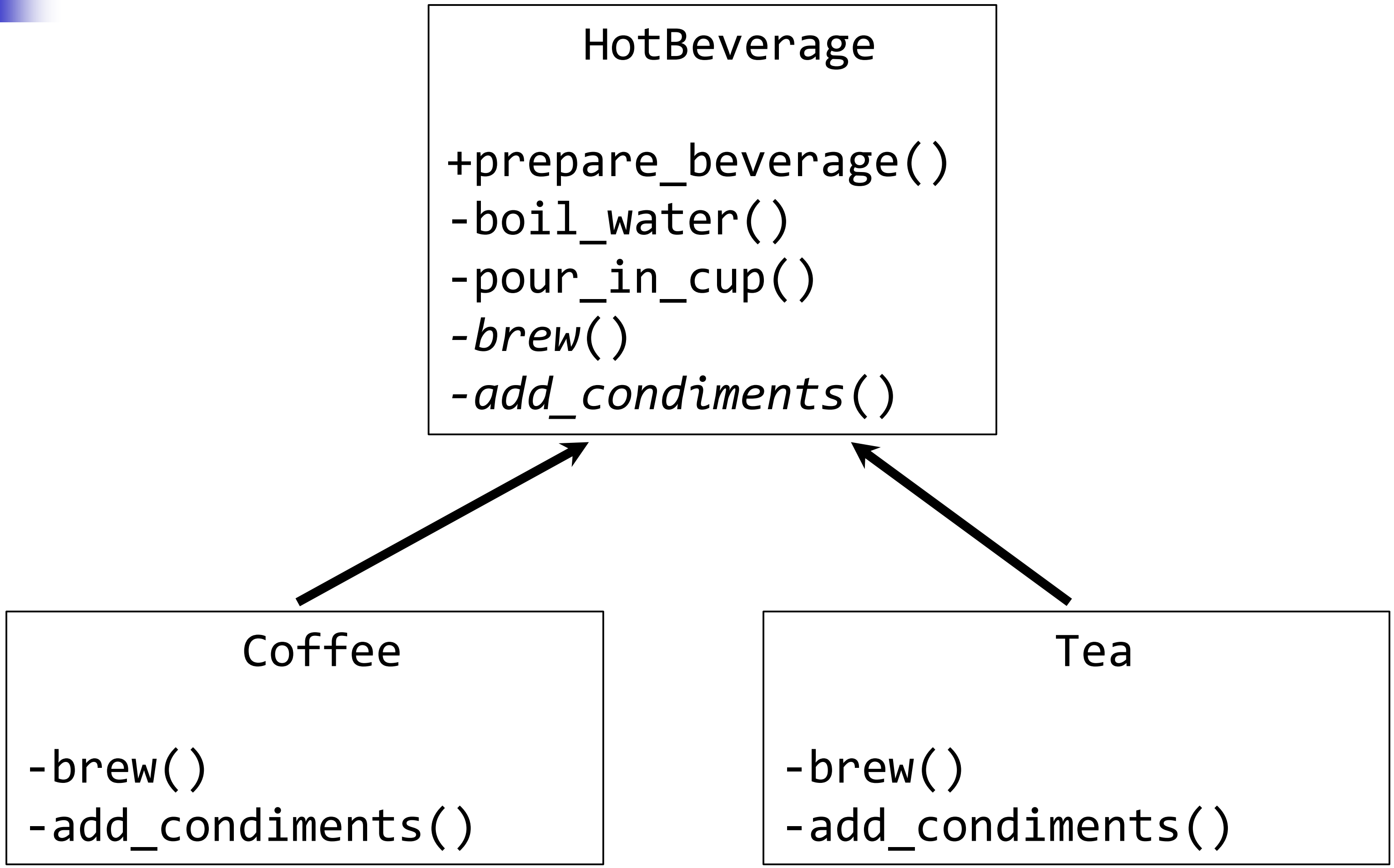
Tea

+prepare_beverage()
-steep_tea_bags()
-add_honey_and_lemon()

Not Really!!!

- `brew_coffee_grinds()` and `steep_tea_bags()` are the same – they just apply to different beverages
- Since these two functions are analogous, make a new pure virtual function *`brew`*`()` in the base class and let derived classes provide their own implementations
- Likewise, since functions for adding milk and lemon are also analogous, make a new pure virtual function *`add_condiments`*`()` and let derived classes provide their own implementations

Newly Redesigned Classes



HotBeverage

+prepare_beverage()
-boil_water()
-pour_in_cup()
-brew()
-add_condiments()

Coffee

-brew()
-add_condiments()

Tea

-brew()
-add_condiments()

prepare_recipe()

```
void HotBeverage::prepare_recipe()  {
 boil_water()
 brew()
 pour_in_cup()
 add_condiments()
}
```

Design Patterns

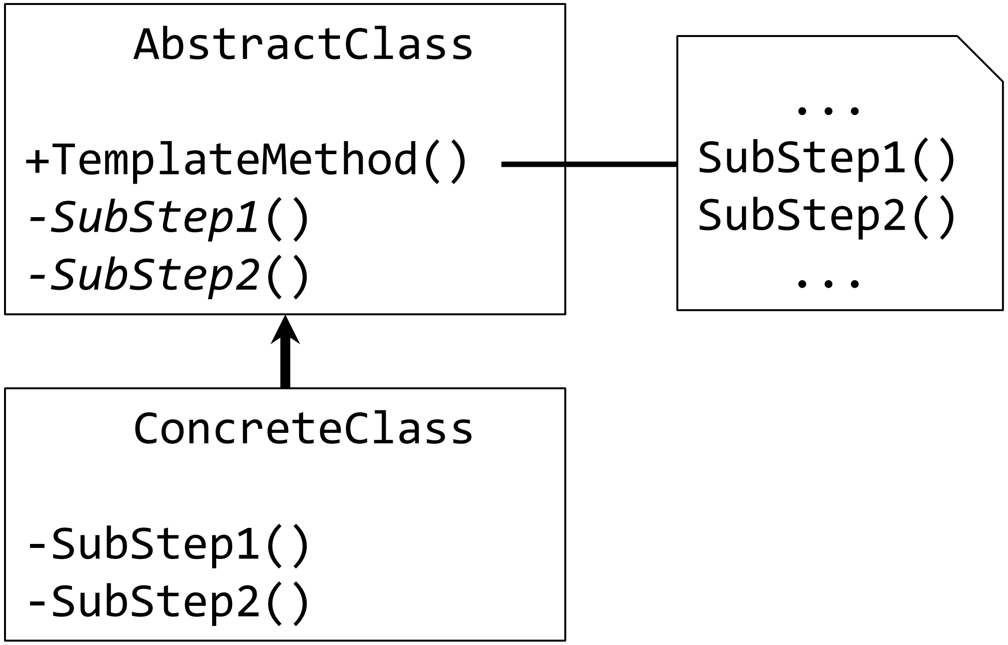
- Design patterns allow us to provide:
  - A high-level perspective on the problem and
  - A high-level perspective on the process of design and object orientation
- That is, patterns help you see the forest and the trees
- This allows us to:
  - Reuse solutions because we don’t need to reinvent solutions to commonly occurring problems (improved modifiability and maintainability of code)
  - Establish common terminology by providing a common point of reference during the project’s analysis and design phase (improved team communications and individual learning)

Template Method Pattern (1/2)

- Intent:
  - Defines an algorithm's skeleton
  - Defers implementation of one or more of these steps to derived classes
- Problem: There is a algorithm or set of steps to follow that is consistent at one level of detail, but individual steps may have different implementations at a lower level of detail
- Solution: Allow for definition of substeps that vary while maintaining a consistent basic process

Template Method Pattern (2/2)

Implementation: Create an abstract class that defines a non-virtual function that implements the steps of the algorithm. Concrete classes provide implementations of certain steps in the algorithm.



Typical Problem

- Software needs to support systems for many (> 100) different companies
  - Rules are (more or less) similar
  - Always subtle differences
- Code becomes increasingly hard to maintain:
  - Many "if-then-else" statements scattered throughout to check which situation was current and to handle it

Typical Solutions

- Continue adding more “if-then-else” statements
  - Common approach
  - Difficulty is with switch creep
  - A few “if-then-else” statements/switches manageable
  - However, at some point, code becomes difficult to read and understand
- Copy and paste the code for each case
  - Results in duplication
  - Advantage is at least each section is clear because it only relates to one situation
- Neither alternative is good

Third Alternative

- Template Method pattern offers a third alternative
- First, let’s see what happens when copy-and-paste approach is used to update code
- Second, after this duplication of process is recognized, how one can refactor the code to eliminate it

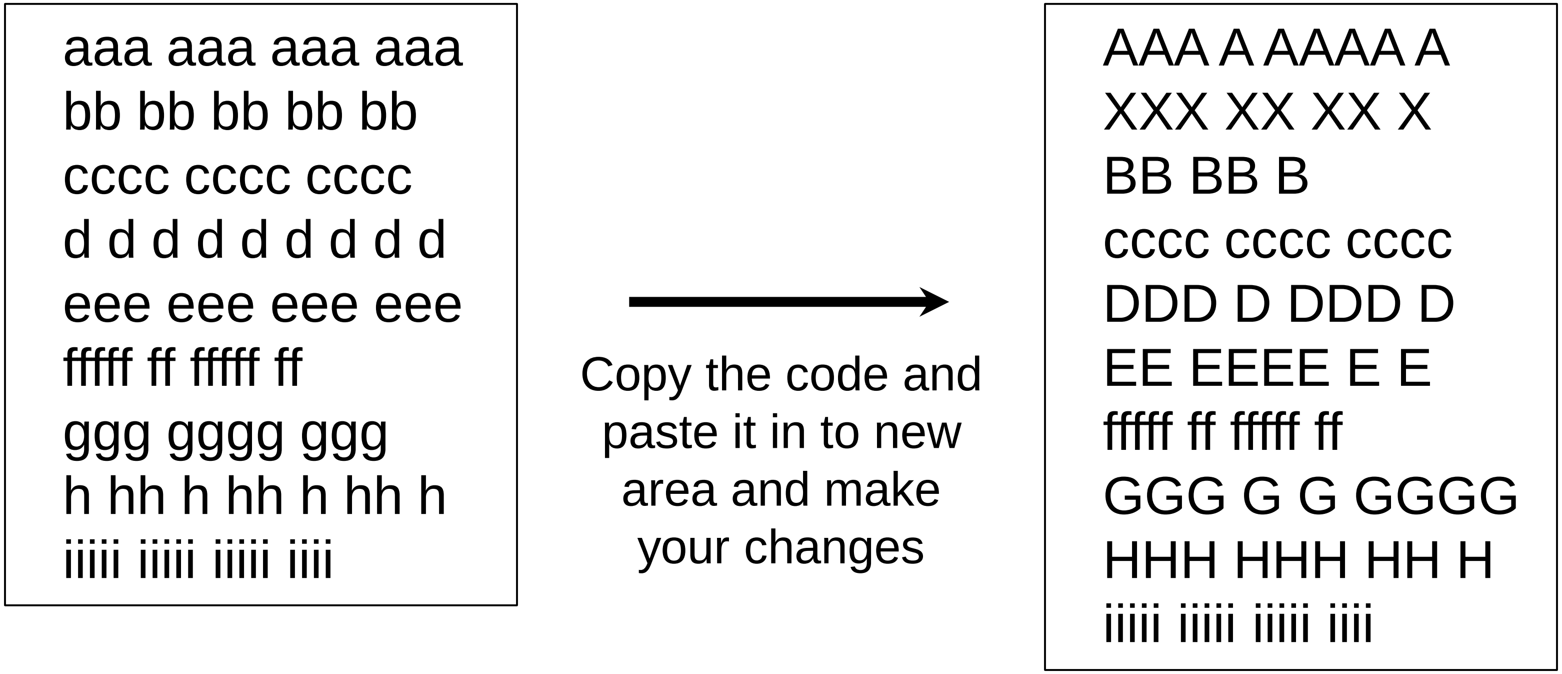
Original Code

MyClass

```
some_method() {
  aaa aaa aaa aaa
  bb bb bb bb bb
  cccc cccc cccc
  d d d d d d d d d
  eee eee eee eee
  fffff ff fffff ff
  ggg gggg ggg
  h hh h hh h hh h
  iiiii iiiii iiiii iiii
}
```

Original Code And New Code

Use copy-and-paste approach to create new code from existing code results in redundancies



aaa aaa aaa aaa
bb bb bb bb bb
cccc cccc cccc
d d d d d d d d d
eee eee eee eee
fffff ff fffff ff
ggg gggg ggg
h hh h hh h hh h
iiiii iiiii iiiii iiii

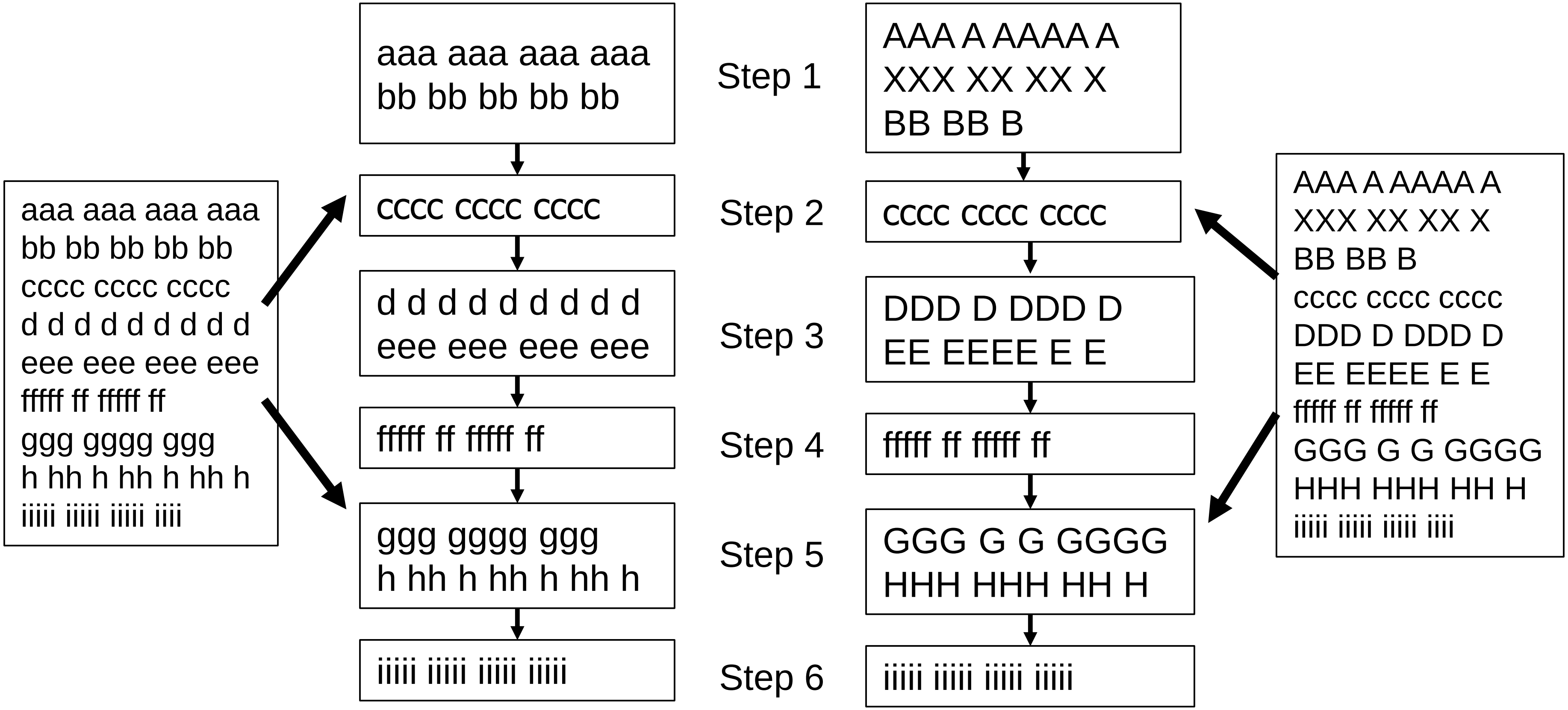
Copy the code and paste it in to new area and make your changes

AAA A AAAA A
XXX XX XX X
BB BB B
cccc cccc cccc
DDD D DDD D
EE EEEE E E
fffff ff fffff ff
GGG G G GGGG
HHH HHH HH H
iiiii iiiii iiiii iiii

Two Types of Duplication

- Obvious duplication
  - Lines with c’s, f’s, and i’s, are duplicated code
  - This is redundant code when we copied and pasted original code
- Another duplication is sequence of operations common to both code fragments
  - Well defined sequence of steps but implementation of some of steps has changed

Comparing Code to Identify Redundancies



aaa aaa aaa aaa
bb bb bb bb bb
cccc cccc cccc
d d d d d d d d d
eee eee eee eee
fffff ff fffff ff
ggg gggg ggg
h hh h hh h hh h
iiiii iiiii iiiii iiii

aaa aaa aaa aaa
bb bb bb bb bb

Step 1

AAA A AAAA A
XXX XX XX X
BB BB B

cccc cccc cccc

Step 2

cccc cccc cccc

d d d d d d d d d
eee eee eee eee

Step 3

DDD D DDD D
EE EEEE E E

fffff ff fffff ff

Step 4

fffff ff fffff ff

ggg gggg ggg
h hh h hh h hh h

Step 5

GGG G G GGGG
HHH HHH HH H

iiiii iiiii iiiii iiiii

Step 6

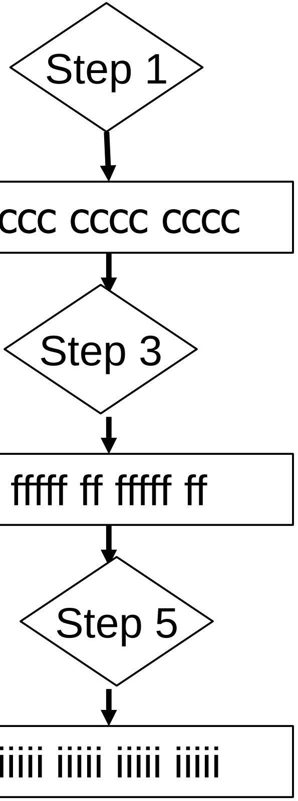
iiiii iiiii iiiii iiiii

AAA A AAAA A
XXX XX XX X
BB BB B
cccc cccc cccc
DDD D DDD D
EE EEEE E E
fffff ff fffff ff
GGG G G GGGG
HHH HHH HH H
iiiii iiiii iiiii iiii

Eliminating Duplication

- Template Method pattern could be used to eliminate duplication:
  - Prescribe a base class that implements the step sequence
  - Each case then has its own derivative class to implement the specified steps

Simpler ‘Template’



Each diamond is candidate for separate methods