Szef Kuchni

W tym zadaniu wcielisz się w rolę szefa kuchni, który za pomocą wzorca Dekorator przygotuje posiłek z wybranych składników.

W tym celu należy najpierw utworzyć abstrakcyjną klasę o nazwie `Meal`. Posiadać ona będzie jedną publiczną metodę o nazwie `prepareMeal` :

```
public void prepareMeal() {
    System.out.println("Przygotowuję danie.");
}
```

Kolejnym krokiem będzie dodanie klas bazowych posiłku. Będą to klasy `PotatoMeal` oraz `RiceMeal`. Każda z nich powinna rozszerzać klasę abstrakcyjną oraz nadpisywać metodę `prepareMeal`, zmieniając wyświetlany komunikat na taki, który będzie odpowiadał bazie naszego posiłku.

Na przykład:

```
@Override
public void prepareMeal() {
    System.out.println("Przygotowuję danie na bazie ziemniaków.");
}
```

Następnie trzeba będzie utworzyć abstrakcyjną klasę dekoratora o nazwie `MealDecorator`. Klasa ta również będzie rozszerzać

klasę abstrakcyjną `Meal`. Poza tym będzie posiadać jedno pole z dostępem domyślnym typu `Meal` o tej samej nazwie.

Ponadto klasa ta powinna posiadać konstruktor z argumentem w postaci przekazywanego obiektu typu `Meal` i nadpisywać metodę `prepareMeal` :

```
@Override
public void prepareMeal() {
    this.meal.prepareMeal();
}
```

Teraz pozostało jeszcze utworzenie konkretnych klas dekorujących, które będą rozszerzać klasę `MealDecorator`. Niech będą to klasy o nazwach `ChickenMealDecorator`, `ShrimpMealDecorator` or az `SauceMealDecorator`.

Każda z tych klas powinna posiadać konstruktor, do którego przekazywany będzie obiekt typu `Meal`. W tym konstruktorze będziemy wywoływali konstruktor rodzica, do którego przekażemy przekazany nam obiekt:

```
public ChickenMealDecorator(Meal decoratedMeal) {
   super(decoratedMeal);
}
```

W każdej z konkretnych klas dekorujących trzeba zaimplementować metodę, która będzie wyświetlała komunikat informujący o dodaniu konkretnego składnika do dania. Metody te

będą nazywać się
odpowiednio: `addChicken`, `addShrimp` oraz `addSauce`.

Poza tym w każdej z tych klas należy nadpisać
metodę `prepareMeal`, w której wywołujemy tę samą metodę z klasy rodzica oraz jedną z metod wspomnianych powyżej.

Przykład implementacji z klasy `ChickenMealDecorator`:

```
@Override
public void prepareMeal() {
    meal.prepareMeal();
    addChicken();
}
private void addChicken() {
    System.out.println("Do dania dodaję kurczaka.");
}
```

Teraz będzie można przejść nareszcie do wywołania kodu w metodzie `main`:

```
System.out.println("Nowy posiłek: ");
Meal riceMeal = new RiceMeal();
riceMeal.prepareMeal();
System.out.println("\nNowy posiłek: ");
Meal riceMealWithShrimp = new ShrimpMealDecorator(new
RiceMeal());
riceMealWithShrimp.prepareMeal();
System.out.println("\nNowy posiłek: ");
Meal potatoMealWithChickenAndSauce = new
SauceMealDecorator(new ChickenMealDecorator(new
PotatoMeal()));
potatoMealWithChickenAndSauce.prepareMeal();
```

Jak widać dzięki poprawnemu zaimplementowaniu wzorca, możemy mieć dowolną kombinację składników oraz taką ich ilość, jaką chcemy.

Stałym elementem jest to, że wszystkie obiekty będą ostatecznie typu `Meal` oraz to, że będziemy na nich w stanie wywołać metodę `prepareMeal`.

Efektem tego kodu powinny być komunikaty podobne do tych poniżej:

```
Nowy posiłek:
Przygotowuję danie na bazie ryżu.
Nowy posiłek:
Przygotowuję danie na bazie ryżu.
Do dania dodaję krewetki.
Nowy posiłek:
Przygotowuję danie na bazie ziemniaków.
Do dania dodaję kurczaka.
Danie polewam sosem.
```