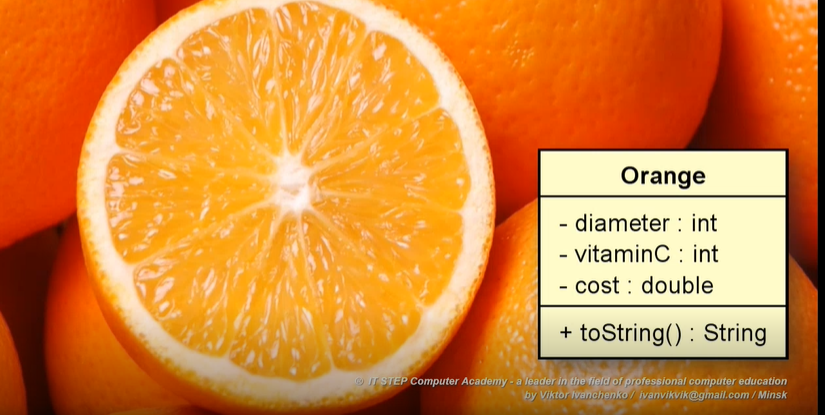
MegaShopProject реализация:

Магазин продуктов: товары- апельсин, молоко, хлеб.

Создаем entity-classes. Выделяем характеристики классов, исходя их важности в данной предметной области

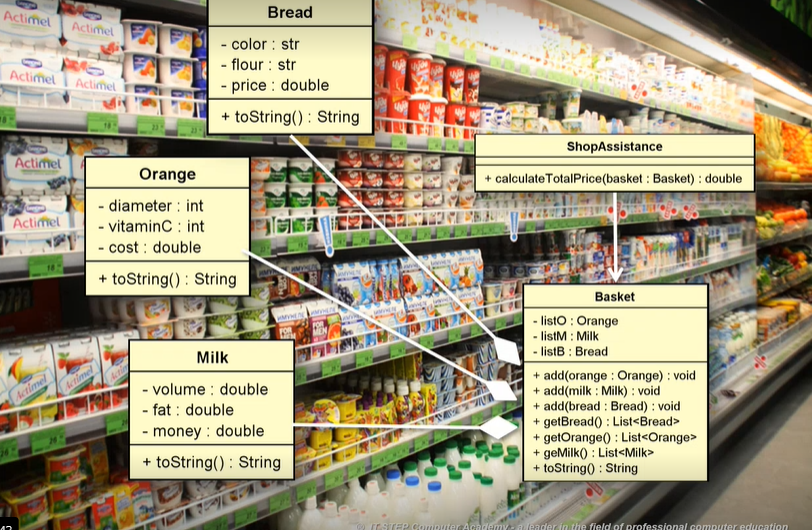




Т.ж. необходимо создать класс-контейнер, Basket, которая будет агрегировать в себе нужное количество продуктов



Исходя из принципа единственной ответственности, должна быть только одна причина для изменений класса, корзина будет только содержать в себе объекты и предоставлять интерфейс для удаления и добавления товаров(рекомендация: сначала реализовать на базе массива, а после прохождения коллекций, провести рефакторинг), а для операций с ними(н-р: подсчета стоимости) ,будет реализована в функциональном классе (бизнес-логика) ShopAssistence()

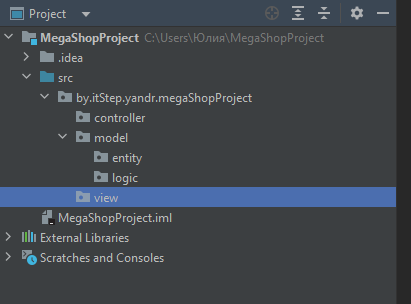


Контейнеризация, в виде композиции

Ассоциация

В данной системе самостоятельными (повторно используемыми) могут быть только классы-сущности, а контейнер и бизнес-логика слишком зависят от классов –сущностей

Создаем архитектуру проекта:



Cоздаем первый класс- сущность, инкапсулируем поля, при создании такого класса должны быть обязательно реализованы 2 конструктора: дефолтный и с параметрами(заполняет все содержимое), при отсутствии дефолтного конструктора, будут проблемы при создании дочерних классов при проведении сереализации (авто) обязательно требуется д.к., если есть какие то дефолтные параметры их сразу в нем и реализуем, для автогенерации getters&setters, toString и конструктора используем Alt+Insert

package by.itStep.yandr.megaShopProject.model.entity;  
  
public class Orange {  
 private int diameter;  
 private int vitaminC;  
 private double cost;  
   
 public Orange(){ дефолтный конструктор  
  
 }  
 public Orange(int diameter, int vitaminC, double cost) {  
 this.diameter = diameter;  
 this.vitaminC = vitaminC; Конструктор, инициализирующий все поля  
 this.cost = cost;  
 }  
  
 public int getDiameter() {  
 return diameter;  
 }  
  
 public void setDiameter(int diameter) {  
 this.diameter = diameter;  
 }  
  
 public int getVitaminC() {  
 return vitaminC;  
 }  
 Геттеры, Сеттеры -   
 public void setVitaminC(int vitaminC) {  
 this.vitaminC = vitaminC; возможность доступа к инкапсу-  
 } лированным полям  
  
 public double getCost() {  
 return cost;  
 }  
  
 public void setCost(double cost) {  
 this.cost = cost;  
 }

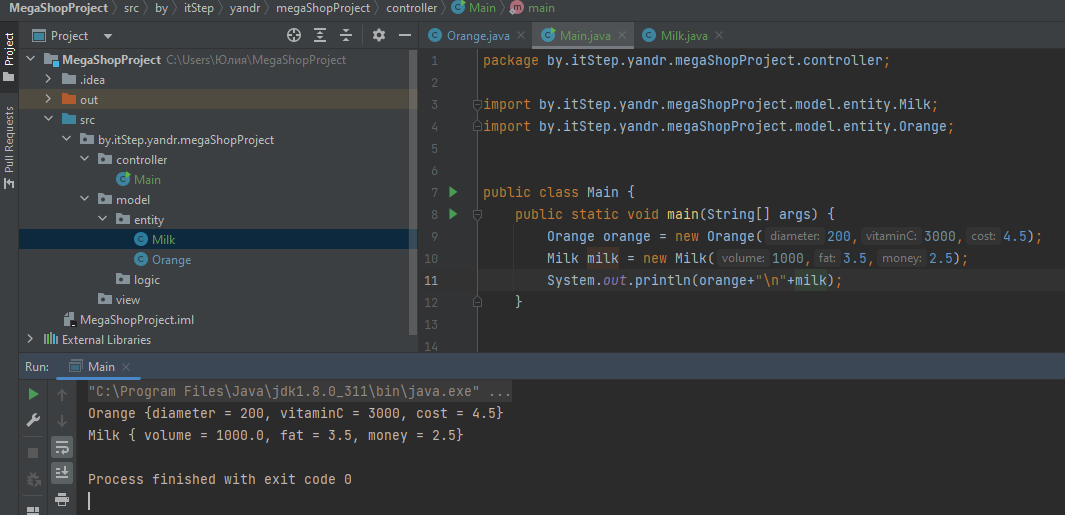
@Override  
 public String toString() { Реализация строкового представления класса  
 return "Orange {" +  
 "diameter = " + diameter +  
 ", vitaminC = " + vitaminC +  
 ", cost = " + cost +  
 "}";  
 }

}

Создаем остальные сущности по тому же принципу

package by.itStep.yandr.megaShopProject.model.entity;  
  
public class Milk {  
 private double volume;  
 private double fat;  
 private double money;

public Milk() {

 }  
  
 public Milk(double volume, double fat, double money) {  
 this.volume = volume;  
 this.fat = fat;  
 this.money = money;  
 }  
  
 public double getVolume() {  
 return volume;  
 }  
  
 public void setVolume(double volume) {  
 this.volume = volume;  
 }  
  
 public double getFat() {  
 return fat;  
 }  
  
 public void setFat(double fat) {  
 this.fat = fat;  
 }  
  
 public double getMoney() {  
 return money;  
 }  
  
 public void setMoney(double money) {  
 this.money = money;  
 }  
  
 @Override  
 public String toString() {  
 return "Milk { " +  
 "volume = " + volume +  
 ", fat = " + fat +  
 ", money = " + money +  
 "}";  
 }  
}

Тестируем сущности

Создаем класс контейнер, реализуем его как динамическую корзину, с использованием массивов

public class Basket {  
 private Orange[] oranges;  
 private Milk[] milks ;   
  
 public Basket(){  
 oranges = new Orange[0];  
 milks = new Milk[0];  
 }  
 public Basket(Orange[] oranges, Milk[] milks) {  
 if (oranges == null) {  
 oranges = new Orange[0];  
 }else{  
 this.oranges = oranges;  
 }  
 if (milks == null) {   
 milks = new Milk[0];   
 }else{  
 this.milks = milks;  
 }  
 }  
 public void add(Orange orange){  
 Orange[]temp = new Orange[oranges.length+1];  
 int i = 0;  
 for (; i < oranges.length ; i++) {  
 temp[i] = oranges[i];  
 }  
 temp[i] = orange;  
 oranges = temp;  
 }  
 public void add(Milk milk){  
 Milk[]temp = new Milk[milks.length+1];  
 int i = 0;  
 for (; i < milks.length ; i++) {  
 temp[i] = milks[i];  
 }  
 temp[i] = milk;   
 milks = temp;  
 }  
 public void removeOrange(int index) {  
 Orange[] temp = new Orange[oranges.length - 1];  
 for (int i = 0, j = 0; i < oranges.length; i++) {  
 if (i != index) {  
 temp[j++] = oranges[i];  
 }  
 }  
 oranges = temp;  
 }  
 public void removeMilk(int index) {  
 Milk[] temp = new Milk[oranges.length - 1];  
 for (int i = 0, j = 0; i < milks.length; i++) {  
 if (i != index) {  
 temp[j++] = milks[i];  
 }  
 }  
 milks = temp;  
 }  
 @Override  
 public String toString() {  
 return "Basket { " +  
 "\noranges = " + Arrays.*toString*(oranges) +  
 "\n, milks = " + Arrays.*toString*(milks) +"}";   
 }  
}

конструктор с параметрами, который принимает уже готовые массивы,  
принимая их, как значения полей данного объекта + активная защита от null

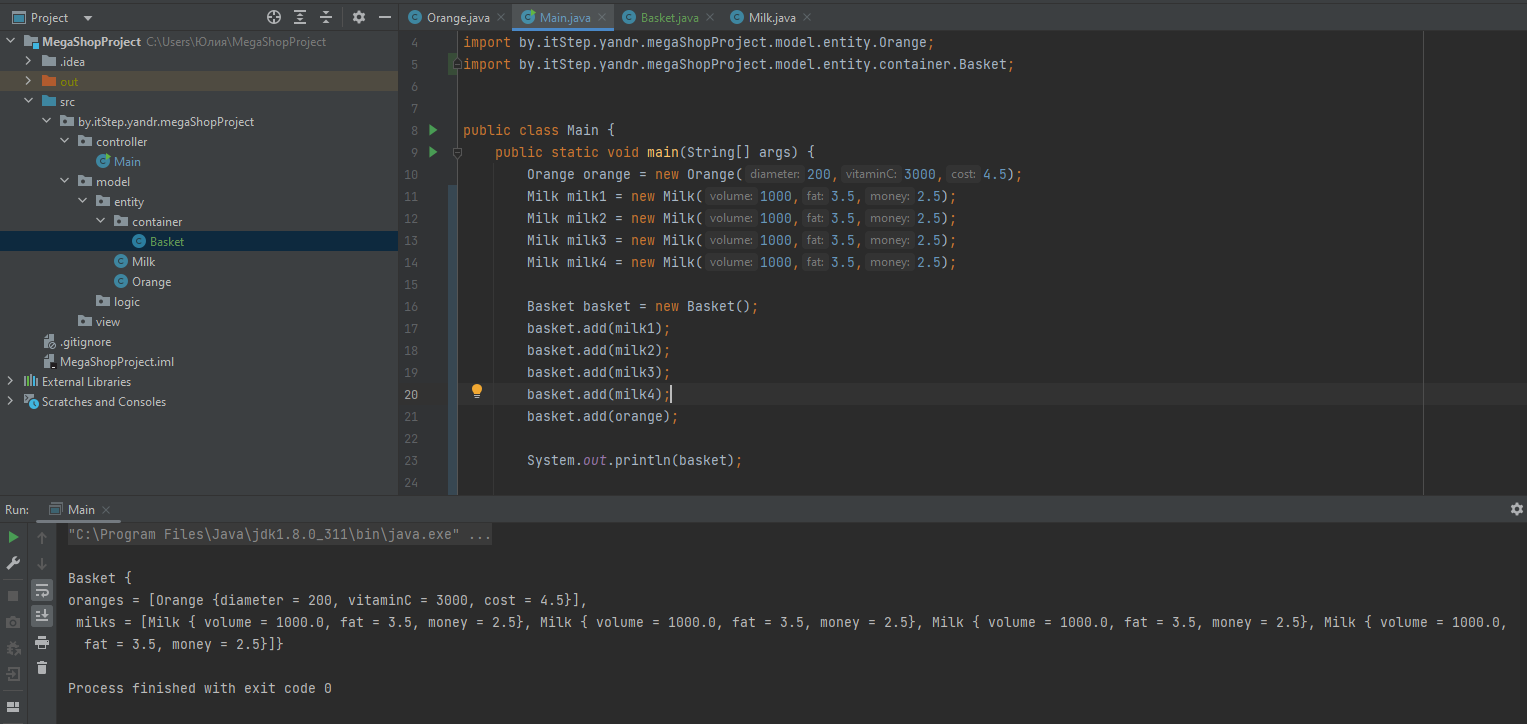
для создания динамического контейнера, в дефолтном конструкторе инициализируем ссылки пустыми объектами, которым при приближении к реальной системе понадобится присвоить дефолтные значения

Методы для добавления и удаления элементов из корзины, реализуются через создание скорректированной соответствующим образом копии и присвоению ее значения в последующем корзине

Строковое представление класса

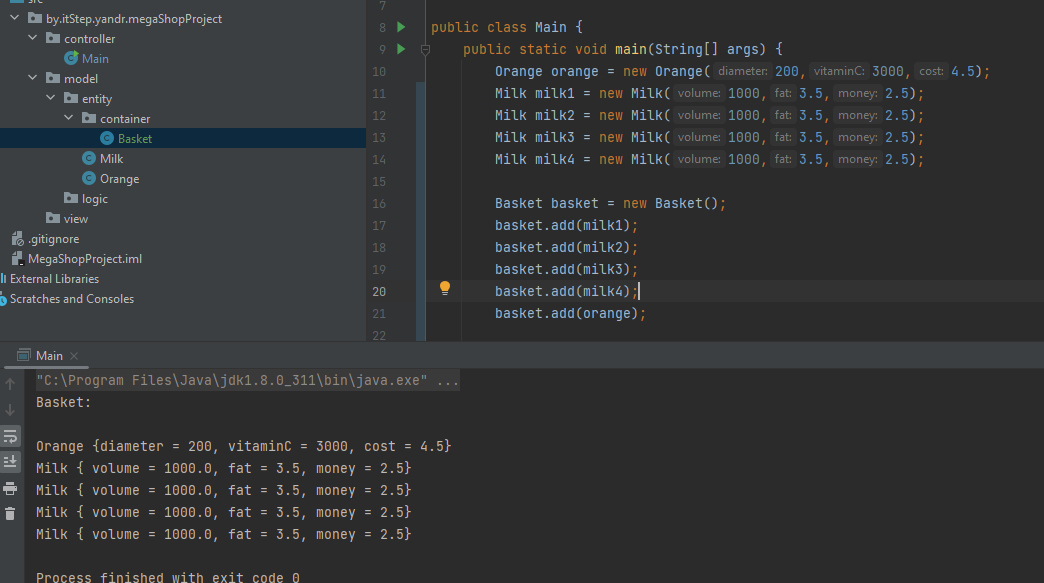
Рефакторинг конструктора с параметрами:

public Basket(Orange[] oranges, Milk[] milks) {  
 this(); вызываем дефолтный конструктор, для инициализации(super() &   
 if (oranges != null) { this()),м.б. только в первой строке  
 this.oranges = oranges;   
 } пере инициализируем, если !=0  
 if (milks != null) {  
 this.milks = milks;  
 }  
}

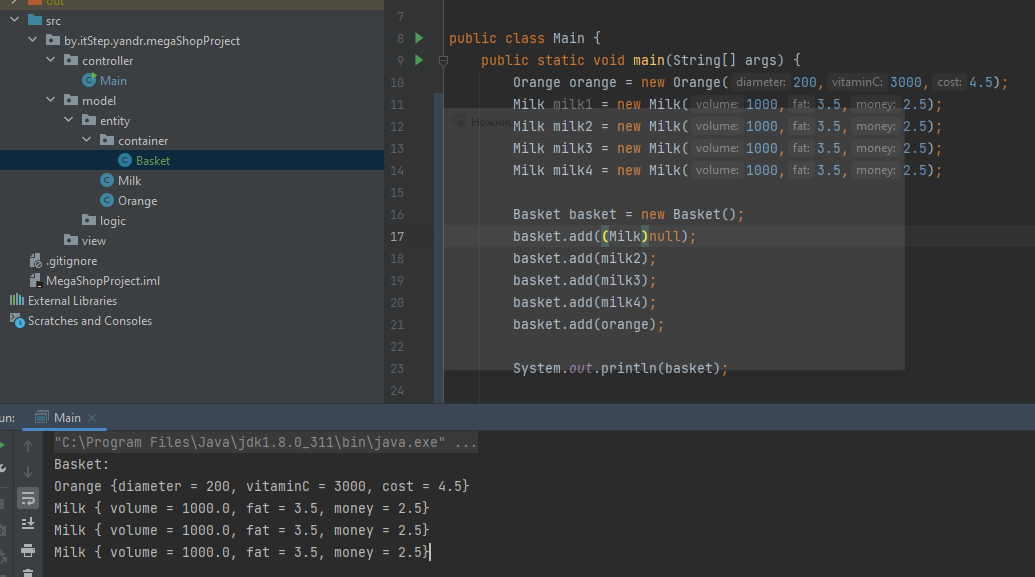
Тестируем корзину:

Рефакторинг toString():

@Override  
 public String toString() {  
 StringBuilder builder = new StringBuilder();  
 if (oranges == null || milks == null  
 || (milks.length == 0 && oranges.length == 0)) {  
 builder.append("Basket is empty.");  
 } else {  
 builder.append("Basket:\n");  
 for (Orange orange : oranges) {  
 builder.append("\n").append(orange);  
 }  
 for (Milk milk : milks) {  
 builder.append("\n").append(milk);  
 }  
 }  
 return builder.toString();  
 }  
}



public void add(Orange orange) {  
 if (orange==null){  
 return;  
 }  
 Orange[] temp = new Orange[oranges.length + 1];  
 int i = 0;  
 for (; i < oranges.length; i++) {  
 temp[i] = oranges[i];  
 }  
 temp[i] = orange; Блокируем null Objects в add  
 oranges = temp;  
}  
  
public void add(Milk milk) {  
 if (milk==null){  
 return;  
 }  
 Milk[] temp = new Milk[milks.length + 1];  
 int i = 0;  
 for (; i < milks.length; i++) {  
 temp[i] = milks[i];  
 }  
 temp[i] = milk;  
 milks = temp;  
}



Добавляем бизнес-логику : class ShopAssistent+ method calculateTotalPrice(). Начинаем с прототипа:

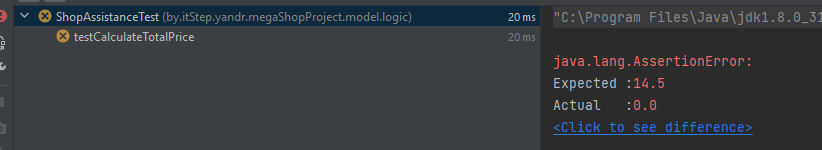
package by.itStep.yandr.megaShopProject.model.logic;  
  
import by.itStep.yandr.megaShopProject.model.entity.container.Basket;  
  
public class ShopAssistance {

Скрываем конструктор (чаще всего для функциональных классов),запрет создания объектов этого класса

private ShopAssistance() {}

public static double calculateTotalPrice(Basket basket){  
 double total = 0;  
 return total;  
 }  
 }

Теперь пишем тест, который заведомо проваливается

public class ShopAssistanceTest {  
 @Test  
 public void testCalculateTotalPrice(){   
 Orange orange = new Orange(200,3000,4.5);  
 Milk milk1 = new Milk(1000,3.5,2.5);  
 Milk milk2 = new Milk(1000,3.5,2.5);  
 Milk milk3 = new Milk(1000,3.5,2.5);  
 Milk milk4 = new Milk(1000,3.5,2.5);  
  
 Basket basket = new Basket();  
 basket.add(milk1);  
 basket.add(milk2);  
 basket.add(milk3);  
 basket.add(milk4);  
 basket.add(orange);  
 double expected = 14.5;  
 double actual = ShopAssistance.*calculateTotalPrice*(basket);  
 *assertEquals*(expected,actual,0.0);  
 }  
}

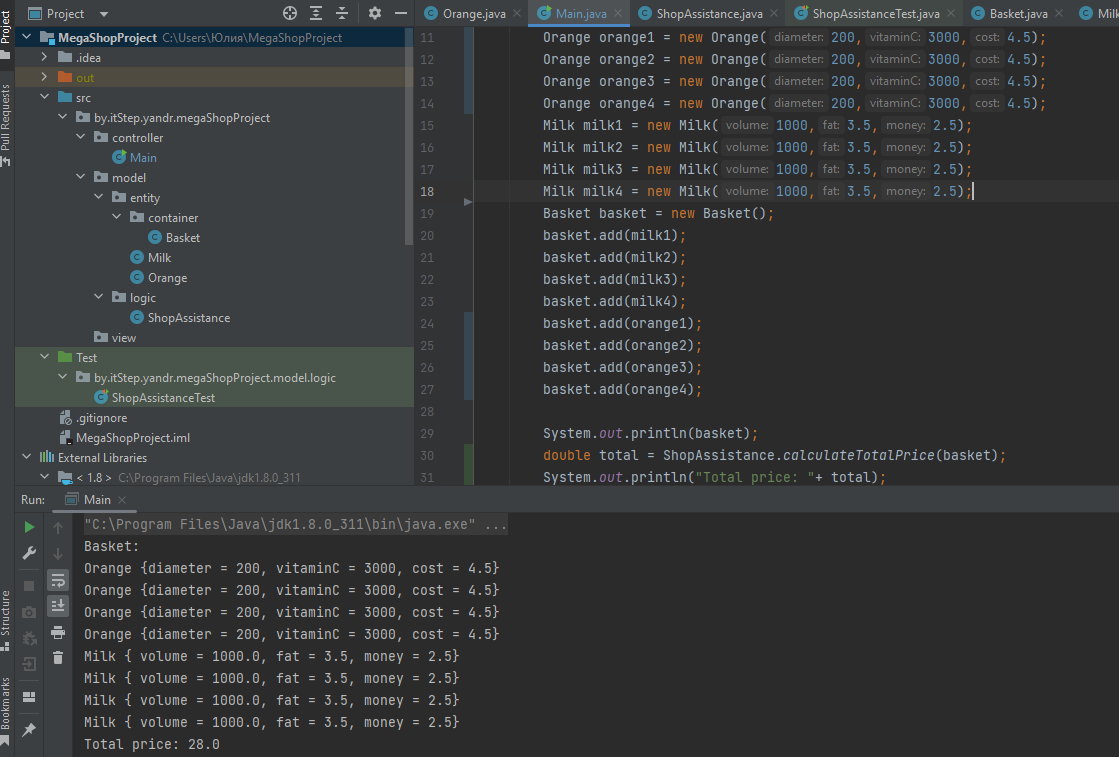
Однако для создания логики калькуляции цены не хватает доступа к содержимому корзины,🡪 добавляем методы для определения размера массива элементов, те их количества getSize() и метод индексного доступа к элементам корзины getOrange(Milk)()

public int getOrangesCount() {  
 return oranges.length;  
}  
  
public int getMilksCount() {  
 return milks.length;  
}  
  
//!!!work-flow  
public Orange getOrange(int index) {  
 return oranges[index];  
}  
  
//!!!work-flow  
public Milk getMilk(int index) {  
 return milks[index];  
}

Теперь можем реализовать калькуляцию : проходимся по каждому элементу контейнеров basket, вытаскиваем на них стоимость и суммируем в total

public static double calculateTotalPrice(Basket basket) {  
 double total = 0;  
 for (int i = 0; i < basket.getOrangesCount(); i++) {  
 total += basket.getOrange(i).getCost();  
 }  
 for (int i = 0; i < basket.getMilksCount(); i++) {  
 total += basket.getMilk(i).getMoney();  
 }  
 return total;  
}

Запускаем в контроллере:



Но при попытке расширить систему введением нового продукта, нам приходится изменять существующий код. Создаем еще одну сущность – хлеб

package by.itStep.yandr.megaShopProject.model.entity;  
  
public class Bread {  
 private String color;  
 private String flour;  
 private double price;  
  
 public Bread() {  
 }  
  
 public Bread(String color, String flour, double price) {  
 this.color = color;  
 this.flour = flour;  
 this.price = price;  
 }  
   
 public String getColor() {  
 return color;  
 }  
  
 public void setColor(String color) {  
 this.color = color;  
 }  
  
 public String getFlour() {  
 return flour;  
 }  
  
 public void setFlour(String flour) {  
 this.flour = flour;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
  
  
 @Override  
 public String toString() {  
 return "Bread { " +  
 "color= " + color + '\'' +  
 ", flour= " + flour + '\'' +  
 ", price= " + price +  
 "}";  
 }  
}

и изменяем существующие классы:

package by.itStep.yandr.megaShopProject.model.entity.container;  
  
import by.itStep.yandr.megaShopProject.model.entity.Bread;  
import by.itStep.yandr.megaShopProject.model.entity.Milk;  
import by.itStep.yandr.megaShopProject.model.entity.Orange;  
  
public class Basket {  
 private Orange[] oranges;  
 private Milk[] milks;  
 private Bread[] breads;  
  
 public Basket() {  
 oranges = new Orange[0];  
 milks = new Milk[0];  
 breads = new Bread[0];  
 }  
  
 public Basket(Orange[] oranges, Milk[] milks,Bread[]breads) {  
 this();  
 if (oranges != null) {  
 this.oranges = oranges;  
 }  
 if (milks != null) {  
 this.milks = milks;  
 }  
 if (breads != null) {  
 this.breads = breads;  
 }  
 }  
  
 public int getOrangesCount() {  
 return oranges.length;  
 }  
  
 public int getMilksCount() {  
 return milks.length;  
 }  
  
 public int getBreadsCount() {  
 return breads.length;  
 }  
  
 //!!!work-flow  
 public Orange getOrange(int index) {  
 return oranges[index];  
 }  
  
 //!!!work-flow  
 public Milk getMilk(int index) {  
 return milks[index];  
 }  
 //!!!work-flow  
 public Bread getBread(int index) {  
 return breads[index];  
 }  
  
 public void add(Orange orange) {  
 if (orange==null){  
 return;  
 }  
 Orange[] temp = new Orange[oranges.length + 1];  
 int i = 0;  
 for (; i < oranges.length; i++) {  
 temp[i] = oranges[i];  
 }  
 temp[i] = orange;  
 oranges = temp;  
 }  
  
 public void add(Milk milk) {  
 if (milk==null){  
 return;  
 }  
 Milk[] temp = new Milk[milks.length + 1];  
 int i = 0;  
 for (; i < milks.length; i++) {  
 temp[i] = milks[i];  
 }  
 temp[i] = milk;  
 milks = temp;  
 }  
 public void add(Bread bread) {  
 if (bread==null){  
 return;  
 }  
 Bread[] temp = new Bread[breads.length + 1];  
 int i = 0;  
 for (; i < breads.length; i++) {  
 temp[i] = breads[i];  
 }  
 temp[i] = bread;  
 breads = temp;  
 }  
  
//!!!  
 public void removeOrange(int index) {  
 Orange[] temp = new Orange[oranges.length - 1];  
 for (int i = 0, j = 0; i < oranges.length; i++) {  
 if (i != index) {  
 temp[j++] = oranges[i];  
 }  
 }  
 oranges = temp;  
 }  
  
///!!!  
 public void removeMilk(int index) {  
 Milk[] temp = new Milk[milks.length - 1];  
 for (int i = 0, j = 0; i < milks.length; i++) {  
 if (i != index) {  
 temp[j++] = milks[i];  
 }  
 }  
 milks = temp;  
 }  
  
 ///!!!  
 public void removeBread(int index) {  
 Bread[] temp = new Bread[breads.length - 1];  
 for (int i = 0, j = 0; i < breads.length; i++) {  
 if (i != index) {  
 temp[j++] = breads[i];  
 }  
 }  
 breads = temp;  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder builder = new StringBuilder();  
 if (oranges == null || milks == null||breads==null  
 || (milks.length == 0 && oranges.length == 0 &&breads.length==0)) {  
 builder.append("Basket is empty.");  
 } else {  
 builder.append("Basket:");  
 for (Orange orange : oranges) {  
 builder.append("\n").append(orange);  
 }  
 for (Milk milk : milks) {  
 builder.append("\n").append(milk);  
 }  
 for (Bread bread : breads) {  
 builder.append("\n").append(bread);  
 }  
 }  
 return builder.toString();  
 }  
}

package by.itStep.yandr.megaShopProject.model.logic;  
  
import by.itStep.yandr.megaShopProject.model.entity.container.Basket;  
  
public class ShopAssistance {  
  
 private ShopAssistance() {  
 }  
  
 public static double calculateTotalPrice(Basket basket) {  
 double total = 0;  
 for (int i = 0; i < basket.getOrangesCount(); i++) {  
 total += basket.getOrange(i).getCost();  
 }  
 for (int i = 0; i < basket.getMilksCount(); i++) {  
 total += basket.getMilk(i).getMoney();  
 }  
 for (int i = 0; i < basket.getBreadsCount(); i++) {  
 total += basket.getBread(i).getPrice();  
 }  
 return total;  
 }  
}

Явное нарушение принципа инверсии зависимости, которое гласит, что:

-Mодули высокого уровня не должны зависеть от модулей низкого уровня; оба должны зависеть от абстракций.

-Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

