**Комментарии к коду**

1. Правило Magic: не использовали числа напрямую в коде, так при работе с массивом не использовано напрямую число – размер массива при написании кода программы;

//Правило Magic: не используй числа напрямую в коде  
Product[] products = {new Product("помидор", 10, "Крым"),  
 new Product("апельсин", 8, "Турция"),  
 new Product("молоко", 3, "Вологда"),  
 new Product("масло", 11, "Краснодар")};

//1. Правило Magic: не используй числа напрямую в коде  
int high = products.length - 1;

2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код

//2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код  
*printProduct*(products);

//2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код  
// печать перечня товаров  
private static void printProduct(Product products[]) {  
 int i = 0;  
 StringBuilder tableHeader = new StringBuilder();  
  
 tableHeader  
 .append("Список товаров:")  
 .append("\n")  
 .append("цена")  
 .append("\t")  
 .append("Наим")  
 .append("\t")  
 .append("производитель");  
 System.*out*.println(tableHeader);  
 for (Product entry : products) {  
 i++;  
 System.*out*.println(entry.toString());  
 }  
}

3. SOLID

3.1 S - принцип единственной ответственности(Single Responsibility Principle) Класс должен выполнять только те функции, для которых он логически предназначен.

Например, в программе классы SortingByPrice, SortingByManufacturersName, SortingByProductName выполняют функции по сортировке массива по стоимости, производителю и названию товара:

public class SortingByPrice implements SortingOfGoods {  
 // сортировка по стоимости товара

public class SortingByManufacturersName {  
 // сортировка по производителю товара

public class SortingByProductName implements SortingOfGoods {  
 // сортировка по наименованию товара

3.2. O - принцип открытости/закрытости(Open Closed Principle). Программные сущности должны быть открыты для расширения, но закрыты для модификации.

Интерфейс SortingOfGoods открыт для расширения (реализация метода quickSort ) , но закрыт для модификации

// O - принцип открытости/закрытости(Open Closed Principle). Программные сущности должны быть открыты для расширения, но закрыты для модификации.  
  
public interface SortingOfGoods {  
 public static void quickSort(Product[] products, int low, int high) {  
  
 }  
}

3.3. L-принцип замены Барбары Лисков(Liskov Substitution Principle). Наследуй только тогда, когда можешь играть роль за предка.

Наследники модифицируют возможности классов своих предков, наследники могут заменить своих предков в местах, где требуется предок. Указанные классы модифицируют интерфейс SortingOfGoods.

public class SortingByProductName implements SortingOfGoods {  
 // сортировка по наименованию товара  
 // L-принцип замены Барбары Лисков(Liskov Substitution Principle)Наследуй только тогда, когда можешь играть роль за предка.  
 public static void quickSort(Product[] products, int low, int high) {

public class SortingByManufacturersName {  
 // сортировка по производителю товара  
 // L-принцип замены Барбары Лисков(Liskov Substitution Principle)Наследуй только тогда, когда можешь играть роль за предка.  
 public static void quickSort(Product[] products, int low, int high) {

public class SortingByPrice implements SortingOfGoods {  
 // сортировка по стоимости товара  
 // L-принцип замены Барбары Лисков(Liskov Substitution Principle)Наследуй только тогда, когда можешь играть роль за предка.  
 protected static void quickSort(Product[] products, int low, int high) {

public interface SortingOfGoods {  
 public static void quickSort(Product[] products, int low, int high) {  
  
 }  
}

3.4. I- принцип сегрегации (разделения) интерфейса(Interface Segregation Principle). Много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения.

Теперь вместо того, чтобы нашей логике в main зависеть от деталей реализации класса Product (в данном случае сортировки перечня продуктов), наша логика тут будет зависеть только от абстракции процесса сортировки (от интерфейса SortingOfGoods).

// I- принцип сегрегации (разделения) интерфейса(Interface Segregation Principle).  
// сортировка по цене  
 System.*out*.println("-----Сортировка по цене-----");  
 SortingByPrice.*quickSort*(products, low, high);  
 //2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код  
 *printProduct*(products);  
  
 System.*out*.println("-----Сортировка по производителю-----");  
 SortingByManufacturersName.*quickSort*(products, low, high);  
 //2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код  
 *printProduct*(products);  
  
 System.*out*.println("-----Сортировка по товару-----");  
 SortingByProductName.*quickSort*(products, low, high);  
 //2. Правило DRY (Don’t Repeat Yourself): не повторяй свой код  
 *printProduct*(products);  
 System.*out*.println("---------------------------------");

public interface SortingOfGoods {  
  
 public static void quickSort(Product[] products, int low, int high) {  
 }  
  
}

public class SortingByProductName implements SortingOfGoods {  
 public static void quickSort(Product[] products, int low, int high) {

public class SortingByManufacturersName {  
  
 public static void quickSort(Product[] products, int low, int high) {

public class SortingByPrice implements SortingOfGoods {  
 protected static void quickSort(Product[] products, int low, int high) {