Javaプログラミング基礎実習

14. 定数

株式会社ジードライブ

今回学ぶこと

- 定数の使い方
- リファクタリングについて
- 列挙型の使い方

定数とは

- 値を変更できない変数のこと
- 変数宣言文の型名の前に **`final`** をつけると、その変数は定数として扱われる
  - 宣言と同時に値を代入する
- 定数の名前は慣例的に大文字とアンダースコアの組み合わせを用いる
  - 例： `MAX_LENGTH`, `HOST_NAME` など

定数の使い方

書式

```
final 型 定数名 = 値;
```

例：定数PI(円周率)の定義

```
private static final double PI = 3.1416;

public static void main(String[] args) {
  double radius = 5;
  double length = 2 * PI * radius;
   …
```

※ static, private, public については後の章で学習する

定数を使うメリット

- プログラム中の数値の意味がよりはっきりする
  - 例：定数を使わない場合

0.1という数値が何を示すのか明確でない

```
double tax = unitPrice * itemCount * 0.1;
```

  - 例：定数を使った場合

税率を掛けていることが分かりやすい

```
double tax = unitPrice * itemCount * TAX_RATE;
```

- 後で値が変更になった際の修正作業が楽になる

```
private static final double TAX_RATE = 0.1;

(…中略…)
double tax = unitPrice1 * itemCount1 * TAX_RATE;
double price = unitPrice2 * itemCount2 * (1 + TAX_RATE);
```

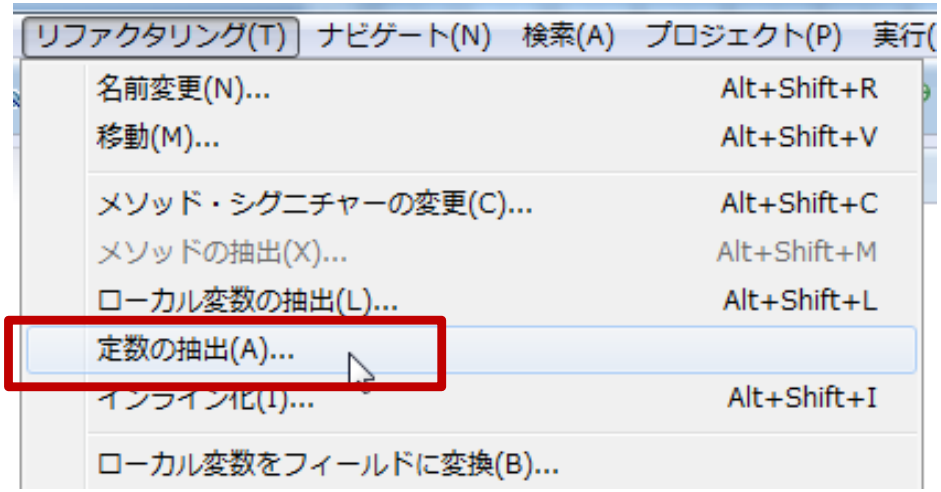
税率が変わった場合はここだけ変更すればよい

リファクタリングについて

- プログラムの処理結果を変えることなく、プログラムコードを変更し改善することをリファクタリング(Refactoring)と呼ぶ
- リファクタリングには様々な観点での作業があるが定数化もそのひとつである
  - クラス名の変更やパッケージの移動もリファクタリングに含まれる

Eclipseのリファクタリング機能

- Eclipseには様々なリファクタリング作業を補佐する機能がある
- 定数に関するリファクタリング機能
  - 「リファクタリング」⇒「定数の抽出」



練習

- 練習14-1

列挙型

- 複数の定数をグループ化して扱える型
  - enum(enumeration:列挙)で定義された定数を列挙子と呼ぶ

書式

```
private enum 型名 {
  値1,
  値2,
  ...
}
```

例

```
private enum CustomerType {
  NORMAL,
  SPECIAL
}
```

列挙型の利用

利用例：メソッドの引数として利用

```
private enum CustomerType {
  NORMAL,
  SPECIAL
}

private static void greeting(CustomerType type) {
  // 列挙型の値は == で比較できる
  if (type == CustomerType.SPECIAL) {
    System.out.println("特別価格でのご案内です");
  } else {
    System.out.println("通常価格でのご案内です");
  }
}
```

列挙型のメリット

例：列挙型を使わない場合のコード

```
private static void greet(int type) {
  switch(type){
    case 0: System.out.println("こんにちは"); break;
    case 1: System.out.println("Hello"); break;
    default: System.out.println("Aloha");
  }
}

public static void main(String[] args) {
  greet(1);
}
```

引数「1」が何を示しているのか不明確

列挙型のメリット

例：列挙型を使用した場合のコード

```
private enum Language {
  JAPANESE,
  ENGLISH,
  HAWAIIAN
}

private static void greet(Language type) {
  switch (type) {
    case JAPANESE: System.out.println("こんにちは"); break;
    case ENGLISH: System.out.println("Hello"); break;
    default: System.out.println("Aloha");
  }
}

public static void main(String[] args) {
  greet(Language.ENGLISH);
}
```

case句では
「**Language.**」は不要

引数が JAPANESE, ENGLISH, HAWAIIANに限定され、
意味も明確になった

練習

- 練習14-2