プログラミングA 第8回・宿題　回答

1　宿題 1

- クラス自身
  - test パッケージ内
- name フィールド
  - test パッケージ内
- デフォルトコンストラクタ
  - Access クラス自身のみ
- 引数を取るコンストラクタ
  - test パッケージ内
- dosome メソッド
  - test パッケージ内

2　宿題 2

- Access2 から、Access の
  - name フィールドにアクセスできる。
  - dosome メソッドにアクセスできる。
- Access3 から、Access の
  - name フィールドにアクセスできない。**クラスを** *public* **にしたうえで** name フィールドの可視性を public または protected に変更すればアクセスできる。
  - *Dosome* **メソッドにアクセスできない。クラスを** *public* **にしたうえで** *dosome* メソッドにアクセスできる

3　宿題 3

参考資料：Data.java

```
public class Data {
  private int number = 0;
  private String text = "TEXT";
  public void setNumber(int number) {
    this.number = number;
  }
  public int getNumber() {
    return number;
  }
  public void setText(String text) {
```

```
    this.text = text;
  }
  public String getText() {
    return text;
  }
}
```

参考資料：DataClient.java

```
c class DataClient {
  public static void main(String[] args) {
    Data d = new Data();
    d.setNumber(100);
    d.setText("NEW");
    System.out.println(d.getNumber());
    System.out.println(d.getText());
  }
}
```

4 宿題 4

- enqueue メソッドの仕組み
  - キューがいっぱいの場合は RuntimeException を投げる。データを格納する位置を表す last の値の次の位置が、データを取り出す位置を表す first の値と等しければキューがいっぱいと判定する。いっぱいでなければ、配列 values の last の位置に渡されたデータを格納する。
  - 続いて last の値を一つ増やす。ただし、その結果が values のサイズを超える場合は、last の値を 0 にすることで、配列の先頭から再度格納するようにする。この計算処理は、last の値を一つ増やした結果に対して values のサイズによる剰余を計算することで実現している。
- dequeue メソッドの仕組み
  - キューが空の場合は RuntimeException を投げる。last と first の値が等しければ空と判定する。空でなければ、配列 values の first の位置のデータを取り出す。
  - 続いて first の値を一つ増やす。ただし、その結果が values のサイズを超える場合は、first の値を 0 にすることで、配列の先頭から再度取り出すようにする。
- フィールド values について、キューのサイズ (キューが扱う要素数の最大値) よりも、一つ大きなサイズの配列としている理由
  - キューと同サイズの配列にすると、last および first の値だけでは、キューがいっぱいの場合と空の場合を区別できなくなってしまうため。

5 宿題 5

(1)
- クラス名やインタフェース名の衝突を防ぐためのグループ分けの仕組みであり、一定以上の開発規模で衝突が起きる可能性がある場合のその防止や、全体をグループ分けして見通しを良くすること

に有効である。また、再配布・再利用の単位ともできる。

(2)
- クラスやインタフェースのアクセス修飾子には厳しいものから順に 無指定, public の 2 種類がある。無指定のクラスやインタフェースは同一パッケージからのみ使える。public のクラスやインタフェースはあらゆるところから使える。
- メソッド、コンストラクタ、フィールドのアクセス修飾子には厳しいものから順に private, 無指定, protected, public の 4 種類がある。
  - private 指定された要素はクラス内でのみ使える。
  - 無指定の要素は同一パッケージからのみ使える。
  - protected の要素は同一パッケージとサブクラスからのみ使える。
  - public の要素は、当該要素を持つクラスやインタフェースが public であればあらゆるところから、無指定であれば同一パッケージから使える。

(2)
- 時間のかかる処理 (例えば通信) を進めながら他の処理を行いたい場合や、複数の処理 (例えば複数ユーザからのアクセス) を偏ることなく概ね公平に進めたいという場合に、同一時点で複数の処理を並行に進める。