情報処理試験のJava

応用情報処理V

2017/1/20

新居雅行

Agenda

- 基本情報処理技術者試験について
- Javaの試験範囲と出題傾向
- 講義で説明していない内容について
- 参考になるサイト
  - http://itsiken.com/
  - http://情報処理試験.jp/index.html

情報処理試験

国家試験

- 経済産業省による認定
- 情報処理に関する技術力の水準を認定

レベルに応じた複数の試験を提供

- ITパスポート試験
- 情報セキュリティマネジメント
- 基本情報技術者試験←本日の説明対象
- 応用情報技術者試験
- 他に9つの高度試験（かなり難関）

受験

- ITパスポートを除き、年に2回の集合試験を受ける

基本情報技術者試験

シラバスに提示された大項目

- 基礎理論、コンピュータシステム、技術要素、開発技術
- プロジェクトマネジメント、サービスマネジメント
- システム戦略、経営戦略、企業と法務

具体的には

- 午前：選択式の問題・80問・150分
- 午後：選択式の問題・13問中7問・150分
- 指定されたプログラミング言語のうちの1つをマスターしておく
- 言語に依存しない問題がいくつも出るが、プログラミングの知識は必要

Javaの位置づけ

- 対象プロググラミング言語の1つ
- 午前にも問題が出るが、午後の1つの選択問題をこなすために必要
- 簡単さで言えば「表計算」と言われている

試験のJavaの範囲

The Java Language Specification, Third Edition（JLS 3.0）

- http://docs.oracle.com/javase/specs/
- つまり、Oracleが公開しているJava言語全部
- 以前はJIS規格とされていたが、現在はオリジナルを試験範囲としている

考え方としては…

- 「言語」としてのJavaの部分が問題範囲と考えて良い
- オブジェクト指向を含む「言語要素」についてはほぼ網羅していないと問題を解くのは困難

問題傾向

問題の難易度

- 他人が作ったプログラムを解読できないといけない
- ただし、詳細な解説があるので、ある意味、それをしっかり読めば良い
- プログラムはそれなりにややこしいが、プログラミングの常識があれば、選択肢を絞れる

API

- 代表的な一部のものだけが問題に登場している
- 普段からプログラミングしていれば自然に覚えている範囲
- 加えて付録でAPIの説明があるので「暗記」は不要

応用情報処理Ｖで学習した範囲

型、変数、配列、式、演算子

繰り返し、条件分岐

クラスの定義とクラスの利用、継承

スタティッククラス、内部クラス

オブジェクト指向について

Javaのプログラミングには必須の概念

クラスを定義する

- データを記録するプロパティ、データ処理を行うメソッドのみが定義できる
- クラスをもとにインスタンス化して、それが実行するという概念
- インスタンス化されたオブジェクトを参照して利用する

自分でクラスを定義する vs 既存のクラスを利用する

- 原則、両方できるようにならないといけない
- 既存クラスの利用では、APIのドキュメントを参照して、そこに用意されているメソッドで必要な処理を組み立てる

継承

既定義のクラスをもとに新たなクラスを定義する

- クラスのプログラムがある／なしは関係ない
- 原則として元のクラス（親クラス）のプロパティやメソッドをすべて引き継ぐが、スコープに注意が必要

```
class a {
  String str;
  int number;
  public void checking(int x){ }
}

class b extends a {
  public void alert(int y){ }
}
```

クラスbでは、プロパティのstrやメソッドのcheckingが使える。
クラスaではメソッドのalertは使えない。

オーバーロード、オーバーライド

```
class a {
  String str;
  int number;
  public void checking(int x){ }
  public void checking(String s){ }
}
```

オーバーロード
引数構成の異なる同一名のメソッドを定義できる

```
class b extends a {
  public void checking(int x){
    super. checking(x);
  }
}
```

オーバーライド
子クラスで、親クラスと同じメソッドを定義できる

ポリモーフィズム

多態性・多相性

- これだけでは意味が分かりにくい
- ソフトウエアのテキスト要素が、実行時に2つもしくはそれ以上の可能な型を表すことができる能力のこと（オブジェクト指向入門第2版方法論・実践、バートランドメイヤー）

つまりは

- クラスaと、それを継承したクラスbがあるとする
- a obj; と定義した変数aに、クラスbのインスタンスを入れることができる。つまり、クラスbはaの機能を全部持っているので、aとして振る舞うことができる

コンストラクタ

クラス名と同名のメソッド

- newで新たにインスタンス化されるときに呼び出される
- オーバーロード可能、つまりさまざまな引数を取れる

コンストラクタに対するルール

- コンストラクタの最初には、引数なしの親クラスのコンストラクタが自動的に呼び出される
- 引数なしのコンストラクタは定義しなくても利用可能だが、コンストラクタを記述すると、引数無しのコンストラクタも書く
- コンストラクタは継承クラスには継承されない
- 親クラスのコンストラクタを呼び出すにはsuper()などと記述
- 自クラスの別のコンストラクタを呼び出すにはthis()などと記述
- コンストラクタから別のコンストラクタを呼び出すには最初に書く

抽象クラス

```
abstract class animal {
  String name;
  abstract int feet();
}
```

未実装のメソッドがあるクラス
インスタンス化はできないが、
継承したクラスでメソッドの実
装を強制できる

```
class human extends animal {
  public int feet() {
    return 2;
  }
}
```

```
class lion extends animal {
  public int feet() {
    return 4;
  }
}
```

インタフェース

```
interface animal {
  abstract int feet();
  abstract int eyes();
}
```

定義すべきメソッドを記述する。
メソッドの処理は記述しない

```
class tiger implements animal {
  public int feet() { return 4; }
  public int eyes() { return 2; }
}

class spider implements animal {
  public int feet() { return 8; }
  public int eyes() { return 8; }
}
```

例外

プログラム中に問題が発生したら即座にドロップアウト

- エラー処理を含むプログラムが非常に記述しやすくなる
- 例外を使わないと、たとえばif文だらけになる

例外を受け取る

- たとえば、JavaのAPIでは、何か問題があれば、例外を発生するように作られている
- 発生した例外を受け取らないといけないような定義も可能。つまり無視できない

例外を発生させる

- メソッド中で例外を発生させると、そのメソッドをすぐに終了して、呼び出し元に戻る
- 呼び出し元で例外を受け取る仕組みがあることが前提

例外

```
class a {
  void checking(int x) {
    try {
      bool n = this.isMinus(x);
    } catch (Exception e) {
      System.out.println(“Wao!”);
    } finally {
    }
  }
  bool isMinus(int x) throws Exception {
    if ( x < 0 ) {
      throw new Exception(“Error”);
    }
  }
}
```

try以降を実行中に例外が発生したら、catchに飛ぶ。

例外はオブジェクトであり、自分で定義もできる

メソッド定義に例外が発生することを予告している。

実際の例外はthrowで発生

スレッド

同時に複数のプログラムが稼働する状況

- 若干飛躍があるが、「スレッド」の話題が出ているときはマルチタスクと考えて良い
- 現実にはCPUの構成やOSの動作などさまざまな要因がある
- 問題は、Javaのプログラムを並列動作をどうすればできるか？

並列時の問題

同期の必要性

- 同時に1つの変数を複数のスレッドが変更しようとすると、正しく変更しない場合が発生する。それを避けるには、変数をあるスレッドが利用するときに、別のスレッドの利用を排除（ロック）する
- Immutableなクラス（String）に対してはロックは不要

スレッドの状態

- スレッドは待機したり、動いていたりする。休眠してロックを解除したり、休眠から復活などさまざまな状態変更に関する仕組みを持っている

デッドロック

- ある状況において、すべてのスレッドが待ちに入ってしまう
- 「食事する哲学者の問題」が有名

Javaのスレッド対応

Threadクラスを継承したクラス

- runメソッドを記述する（名前が決まっている）
- クラスを生成してstartメソッドを呼び出すと、runメソッドを独立したスレッドで動かす

Runnableインタフェースを実装するクラス

- runメソッドを定義、startメソッドで開始は同じ

難しい点

- CPUが速すぎて直列的にしか動いてくれない場合があるなど、実行時の状況は動作環境に依存する
- 究極的には「思った通りの順序には実行されない」可能性を考える必要があり、問題が特定しづらくなる
- 書き込みを伴うプログラムはいきなり難しくなる

匿名内部クラス

```
AnyClass x = new AnyClass() {
    public void living() {
        System.out.println("Yes! Living");
    }
}
x.living();
```

AnyClassを継承した名前のないクラスを作り、そのインスタンスを生成する。
変数xはAnyClassではなく、それを継承したクラスのオブジェクトを参照する。
ここではAnyClassがlivingメソッドを実装していないといけない。

列挙型

```
public class Test {
    public enum Story {
        MOMOTARO, SARU, KIJI, ONI;

        private int power;
        public int getPower()   {return this.power;}
        public void setPower(int p) {this.power = p;}
    }

    public static void main(String[] args) {
        Story.MOMOTARO.setPower(100);
        Story.SARU.power = 99;
        System.out.println(Story.MOMOTARO.ordinal()
            + ":Power=" + Story.MOMOTARO.getPower());
        System.out.println(Story.SARU.ordinal()
            + ":Power=" + Story.SARU.getPower());
        System.out.println(Story.ONI.ordinal()
            + ":Power=" + Story.ONI.getPower());
    }
}
```

出力例

```
0:Power=100
1:Power=99
3:Power=0
```

クラスの定義と同様にプロパティやメソッドなどを記述できるが、名前を列挙すると、その名前でオブジェクトがすでに出来上がっている。

ordinal()で列挙の番号が得られるなど、いくつかのメソッドはenumだと常に利用できる。

データ構造

きわめて重要な話題

- 複雑なデータをいかに記録するかに関わる
- オブジェクトの集合をどのように扱うかを定義する
- 非常に多機能なAPIが用意されている。情報処理試験では必ずしも出てくる話題ではない模様

オブジェクトの集合

- 配列：オブジェクトが順序を伴って記録されている
- 集合：重複した要素がないオブジェクトの集合
- マップ：キーと値のセットを複数記録可能。キーは重複できないが、値は重複できる
- メソッドは概ね共通となっている

テンプレート

配列などの要素の型を定義する

- List<String> stringList = new LinkedList<String>();
- Listは配列の実装の1つ
- 要素にStringしか設定できなくなる

パラメータ指定ができる

- Map<K, V> map = new HashMap<K, V>();
- K、Vにはクラス利用時に決めることができる

過去問題