SJC-P チェックポイント

宣言とアクセス制御

- ・識別子は、'\$'(通貨記号)、'_'(アンダースコア)、で始まってよい。
- ・パブリッククラスを含まないファイルに適用される命名規約はない。
- ・デフォルトとプロテクトメンバーのアクセス相違はサブクラスについてのみ

メンバ	同じパッケージ	サブクラス
デフォルト		×
プロテクト		

- ・インスタンス変数の修飾子

修飾子	インスタンス変数
abstract	×
synchronized	×
native	×
strictfp	×
transient	(インスタンス変数にのみ指定可)
volatile	(インスタンス変数にのみ指定可)

- ・列挙型は、メソッドの内部に宣言できない。
- ・インターフェースのメソッドは暗黙で public なので、以下のような実装は NG

```
interface Inf {
    void m();
}
class Cls implements Inf {
    void m(); // 一見 OK だが、public が必要！
}
```

オブジェクト指向

- ・Java5 では、オーバーライドメソッドの戻り値をサブクラスに指定することができる。(共変戻り値)
- ・ファイナルメソッドはオーバーライドできない。以下の例は NG

```
class Cls1 {
    final void m(){}
}
class Cls2 extends Cls1 {
    void m(){}
}
```

- ・アクセスしているメソッドの数が少ない場合、結合度は低くなる

代入

- ・スタックとヒープ

変数	格納先
----	-----

インスタンス変数、オブジェクト	ヒープ
ローカル変数	スタック

・初期化ブロック

```
class Cls {
    static int x;
    int y;
    static { x = 1; } // 静的初期化ブロック クラスロード時に実行
    { y = 2; } // インスタンス初期化ブロック インスタンス作成時に実行
}
```

- ・インスタンス初期化ブロックは、super() の後 (スーパークラスのコンストラクタ呼出の後) に実行される。
- ・初期化ブロックは、ファイルに記述された順番に実行される。

ボックス化

- ・以下のラッパーオブジェクトがボックス化により生成される場合、基本データ型の値が同じ場合は、「==」となる。
 - ・ Boolean
 - ・ Byte
 - ・ \u0000 ~ \u007f の文字
 - ・ Short、Integer の -128 ~ 127
- ・ ボックス化、拡大変換、Var-Arg の優先度

<優先度 (高)>
拡大変換

ボックス化

Var-Arg
<優先度 (低)>

- ・ int long Long : 拡大変換 ボックス化は NG
- ・ int Integer Number : ボックス化 拡大変換 は OK

演算子

- ・ x *= 2 + 5 の場合、2 + 5 が先に計算される。
- ・ 列挙型の比較には、== または、equals を利用する。
- ・ instanceof では、2 つの別のクラス階層にまたがる比較を行うことはできない (コンパイルエラー)

フロー制御、例外、アサーション

switch 文

- ・ switch 式では、char、byte、short、int、enum が使用できる (enum または、int に拡大変換できる型)
- ・ case には、リテラルまたは、リテラル値が代入されているファイナル変数のみ指定できる。
- ・ case で、int 以外を評価する場合、範囲外の値はコンパイルエラー (byte に対して、128 等)
- ・ case の値に重複がある場合、コンパイルエラー

- switch 文にて、ボックス化を利用できる (case では利用できない)
- default は最後でなくてもよい

ループ

- while ループでは、括弧内で、変数を宣言できない。while(int i=0) は不可
- do ループでは、while の最後にセミコロンが必要

```
do {
    // 処理
} while ( 条件 );
```

- 拡張 for ループでは、最初のオペランドを宣言しなければいけない

```
int[] a = {1,2,3};
```

OK

```
for (int x : a) {}
```

NG

```
int x = 0;
for (x : a) {}
```

アサーション

- 非常に単純なアサーション

```
private void test() {
    assert (y > x); // y > x を仮定
}
```

- 単純なアサーション

```
private void test() {
    assert (y > x): "y is " + y + " x is " + x; // y > x を仮定
}
```

- あるクラスのアサーションを無効にして、他を有効にする

```
java -ea -da:foo.barpackage.Hoge
```

- あるパッケージに対して、アサーションを無効にして、他を有効にする

```
java -ea -da:foo.barpackage...
```

例外

- JVM がシャットダウンされた場合、finally は呼び出されない。

- ・オーバーライドメソッドでは、以下の例外を投げてよい。
 - ・同じ例外
 - ・範囲の狭い例外
 - ・ランタイム例外
 - ・投げない

例外	内容	種類
ArrayIndexOutOfBoundsException	不正なインデックスを使って配列がアクセスされた	Runtime
ClassCastException	オブジェクトを継承関係にないクラスにキャストしようとした	Runtime
IllegalArgumentException	不正な引数、または不適切な引数をメソッドに渡した	Runtime
IllegalStateException	不正または不適切なときにメソッドが呼び出された	Runtime
NullPointerException	オブジェクトが必要な場合に、アプリケーションが null を使おうとするとスロー	Runtime
NumberFormatException	文字列を数値型に変換しようとしたとき、文字列の形式が正しくない	Runtime
AssertionError	宣言が失敗したことを示すためにスロー	Error
ExceptionInInitializerError	static 初期化子で予想外の例外が発生したことを通知	Error
StackOverflowError	アプリケーションでの再帰の回数が多すぎてスタックオーバーフローが起こる場合にスロー	Error
NoClassDefFoundError	クラス定義が見からない場合にスロー	Error

文字列、入出力、書式化、解析

- ・ java.io

クラス	備考
File	
PrintWriter	Java5 では、File、String からコンストラクトできる
FileWriter	
FileReader	
BufferedWriter	

BufferedReader

- ・ ファイルの生成
 - ・ File オブジェクトから、createNewFile() メソッドを呼ぶ
 - ・ Writer、または Stream を作成する。既に存在していない場合、ファイルが生成される

直列化

直列化しようとするクラスのメンバーとして、参照クラスを使用する場合の戦略

ソースコードにアクセスできる
YES: Serializable を実装
NO: final クラスである
NO: サブクラスを作成し、Serializable を実装
YES: transient 修飾子を使用して、直列化をスキップし、writeObject、readObject を利用する。

日付 数値

- ・ Calendar.roll() メソッドは指定フィールドの値を計算するが、上位の値を変更することはない(月を12ヶ月進めても、年は増えない)
- ・ Locale には、2つのコンストラクタ
 - ・ Locale(String 言語コード)
 - ・ Locale(String 言語コード, String 国コード)

ジェネリックスとコレクション

- ・ 配列自体に、多態性を適用した場合で、実行時に型が不一致の場合、ArrayStoreException となる。

```
Base[] a = new Deriv1[2];  
a[0] = new Deriv2(); // ArrayStoreException が投げられる
```

- ・ <> で囲まれた型は、ジェネリックス型、ジェネリックス型が適用された型をベース型

型	多態性
ベース型	
ジェネリックス型	×

```
ArrayList<Number> l = new ArrayList<Integer>(); //NG  
List<Integer> l2 = new ArrayList<Integer>(); //OK
```

- ・ ジェネリックスを利用したベース型の要素格納等には当然多態性を利用できる

```
List<Number> l = new ArrayList<Number>();  
l.add(new Integer(1)); // OK  
l.add(new Long(2)); // OK
```

- ・ 多態性を利用して、統一的に処理を行いたい(基底クラスが提供するメソッドを呼び出したい等)場合は、ワイルドカード + extend を利用する
- ・ 引数型を、List<? extend Clazz> とすることで、IS-A Clazz にパスする任意のジェネリックス型のリストを渡すことができる。

- このリストに要素を追加することはできない。(実際の型が何であるか不明なため)
- 削除操作などは可能。

```
public void m(List<? extends Number> lst) {
    lst.add(new Integer(1)); // 追加は NG
    lst.remove(1);          // 削除は OK
}
```

- 多態性を利用して、基底クラスの参照に、派生クラスを格納したいような場合は、ワイルドカード + super を利用する。
- 引数型を、List<? super Class> とすることで、Cazz IS-A ? にパスする任意のジェネリクス型のリストを渡すことができる。
- 渡された側では、ジェネリクス型に代入互換性があるクラスを追加することができる。

インナークラス

- static メソッドからは、インスタンス化にアウタークラスの参照が必要
- インスタンスメソッドからは、通常と同様にインスタンス化できる。
- インナークラスからアウタークラスの参照は、クラス名 .this

```
public class OuterCls {
    public static void main(String[] args) {
        OuterCls.InnerCls ic = (new OuterCls()).new InnerCls();
    }
    public void instancingInner() {
        InnerCls ic = new InnerCls();
    }
    class InnerCls {
        public void printParent() {
            System.out.println(OuterCls.this);
        }
    }
}
```