Java SE 7

[Java][SJC-P]

自動リソース管理

・close を手動で行う必要がない
・; で複数のリソースを同時に管理
・java.lang.AutoCloseable インターフェースを実装する必要あり
・AutoCloseable は close() のみを持つ Closeable インターフェースの子

例

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class AutomaticResourceManagement {
    public static void main(String[] args) {

        try (BufferedReader reader
                = new BufferedReader(new FileReader(new File("/home/piroto/work/test.txt")));
              BufferedWriter writer
                = new BufferedWriter(new FileWriter(new File("/home/piroto/work/out.txt")))
                ) {

            String line = null;
            while((line = reader.readLine()) != null) {
                writer.write(line);
                writer.write("\n");
            }

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

ダイヤモンド演算子

■これまで

```
Map<String,Integer> numMap = new HashMap<String, Integer>();
```

■ <> (ダイアモンドオペレーター) シンボルを利用することで、すべての型リストを書く必要はない

```
new hashmap() だと、これまで通りコンパイラはタイプセーフ警告を発する
```

例

```
import java.util.HashMap;
import java.util.Map;

public class DiamondOperator {
    public static void main(String[] args) {
        Map<String,Integer> numMap = new HashMap<>();
```

```
    }
}
```

例外ハンドリング

・| オペレータで、複数の例外をキャッチできる

例

```
import java.io.FileNotFoundException;
import java.nio.file.FileAlreadyExistsException;


public class ExceptionHandling {
    public static void main(String[] args) {

        try {
            doSomethingWithMultiException();
        } catch (FileAlreadyExistsException | FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private static void doSomethingWithMultiException()
            throws FileNotFoundException, FileAlreadyExistsException {
    }
}
```

数値リテラル

・数字リテラル _ を入れられる (値としては無視される)
・二進数リテラル 0b で始める

例

```
public class NumericLiteralWIthUnderScore {

    public static void main(String[] args){

        final int million = 1_000_000;
        final int binary = 0b101;

        System.out.println(million);
        System.out.println(binary);
    }
}
```

文字列による Switch

・case ラベルでは、String.equals() で判定される

例

```
public class StringSwitch {
   public static void main(String[] args) {

       String[] status = {"new","old","New","Old"};
       int idx = (int)(Math.random()*10d % status.length);

       switch (status[idx]) {
       case "new":
```

```
                System.out.println("new:" + status[idx]);
                break;
            case "old":
                System.out.println("old:" + status[idx]);
                break;
            default:
                System.out.println("other:" + status[idx]);
                break;
            }

        }

}
```

Path

・Path はファイルパスへの簡易な参照
・java.io.File と同等で追加機能をもつ

例

```
import java.nio.file.Path;
import java.nio.file.Paths;

public class WorkingWitPath {

    public static void main(String[] args) {

        Path path = Paths.get("/home");
        Path source = path.resolve("piroto/work/test.txt");
        System.out.println(source);

        Path dist = source.resolveSibling("out.txt");
        System.out.println(dist);
    }
}
```

ファイル・ディレクトリ変更の検知

・WatchService の作成。このサービスは WatchKey へのキューからなる
・WatchService でモニターしたいディレクトリ、ファイルを登録
・イベントをリッスンするループの開始
・イベントが発生すると、WatchKey がキューに入れられる
・WatchKey を消費し、問い合わせを実行

例

```
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchEvent.Kind;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;

public class FileChangeNotifications {

    public static void main(String[] args) {
        try {
            WatchService watchService = FileSystems.getDefault().newWatchService();
            Path watchDir = Paths.get("/home/piroto/work");
            watchDir.register(watchService,
                    StandardWatchEventKinds.ENTRY_CREATE,
                    StandardWatchEventKinds.ENTRY_MODIFY,
```

```
                StandardWatchEventKinds.ENTRY_DELETE);
            while(true) {
                WatchKey watchKey = watchService.take();
                for (WatchEvent<?> event : watchKey.pollEvents()) {
                    Kind<?> kind = event.kind();
                    System.out.println(event.context().toString() + " is " + kind);
                }
            }

        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

フォークとジョイン

・fork/join フレームワークは、ExecuterService インターフェースの実装
・複数のプロセッサーがある場合に有利

例

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class ForkAndJoin {
    public static void main(String[] args) {
        try {
            // n番目のフィボナッチ数を求める
            Fibonacci fnc = new Fibonacci(15);
            ForkJoinPool pool = new ForkJoinPool();
            pool.invoke(fnc);

            System.out.println("RESULT:" + fnc.get());
        } catch (Exception e){
            e.printStackTrace();
        }
    }


}
/**
 * @see http://docs.oracle.com/javase/jp/7/api/java/util/concurrent/RecursiveTask.html
 */
class Fibonacci extends RecursiveTask<Integer> {
    final int n;

    public Fibonacci(int n) {
        System.out.println(this.toString());
        this.n = n;
    }

    @Override
    protected Integer compute() {
            if (n <= 1) {
                return n;
            }
            Fibonacci f1 = new Fibonacci(n - 1);
            f1.fork();
            Fibonacci f2 = new Fibonacci(n - 2);
            return f2.compute() + f1.join();
    }
}
```

結果

```
info.typea.javase7.nio.Fibonacci@6ab41dbb
info.typea.javase7.nio.Fibonacci@570c16b7
info.typea.javase7.nio.Fibonacci@22fdffb1
                :
info.typea.javase7.nio.Fibonacci@519dcf69
```

```
info.typea.javase7.nio.Fibonacci@4f9c205b
info.typea.javase7.nio.Fibonacci@13105f32
RESULT:610
```