Best practices - Java

Rules	Example
Class name should be nouns, in mixed case with the first letter of each internal word capitalized.	class Date class DateFormat
Interface names should be capitalized like class names	interface OperateCar
Methods should be verbs, in camel case	run(); runFast();
Variable names should not start with underscore or dollar sign. Variables should be in camel case. Variable name should be meaning full.	String totalLength;
Characters of Constants should be uppercase with words separated by underscores	static final int MIN_WIDTH = 6;
Avoid lines longer than 80 characters, since they are not handled well by many terminals and tools.	
When an expression will not fit on a single line,	
• Break after a comma	`someMethod(longExpression1, longExpression2, longExpression3)`
• Break before an operator	`longName1 = longName2 * (longName3 + longName4 - longName5) + 4 * longname6;`

• Align the new line with the previous line, if it is confusing intend 8 spaces before the line.	`private static synchronized horkingLongMethodName(int anArg, Object anotherArg, String yetAnotherArg) {}`
	
	`private static synchronized horkingLongMethodName(int anArg, Object anotherArg, String yetAnotherArg, Object andStillAnother) {}`

Minimize the accessibility of class members (fields) as inaccessible as possible	

Use underscores in numeric letters	`int maxUploadSize = 20_971_520;`
	`long accountBalance = 1_000_000_000L;`
	`float pi = 3.141_592_653_589F;`

Avoid empty catch blocks

Inform the user about the exception

Use Enums or constants instead of constant interface

```
public interface Color {
    public static final int RED = 0xff0000;
    public static final int BLACK = 0x000000;
    public static final int WHITE = 0xffffff;
}
```

Avoid redundant initialization

```
    int number;  // number will have default value: 0
    float ratio;     // default value: 0.0
    boolean success; // default value: false
```

Avoid using for loop with indexes when it can be replaced with enhanced for loop (for each loop)

```
for (type var : array) {
    statements using var;
}
```

Use StringBuilder or StringBuffer instead of String concatenation

Ex:

The following code snippet uses the + operator to build a SQL query:

```
String sql = "Insert Into Users (name, email, pass, address)";
sql += " values ('" + user.getName();
sql += "', '" + user.getEmail();
sql += "', '" + user.getPass();
sql += "', '" + user.getAddress();
sql += "')";
```

With StringBuffer,

```
StringBuilder sbSql = new StringBuilder("Insert Into Users (name, email, pass,
address)");

sbSql.append(" values ('").append(user.getName());
sbSql.append("', '").append(user.getEmail());
sbSql.append("', '").append(user.getPass());
sbSql.append("', '").append(user.getAddress());
sbSql.append("')");

String sql = sbSql.toString();
```

When declaring collection objects, references to the objects should be as generic as possible,

Ex:

```
public class CollectionsRef {

   private Set<Integer> numbers = new ArrayList<String>();

}
```