# Java hints 🔰 Game Camp 2018

This document holds many of the Java programming foundations and secret hints that developers use everyday without knowing.

## Index



## Classes

**ALWAYS**:

- Write import statement(s) at first line of code, outside of the class definition curly braces.
- Use a consistent format when you define a class. For example, define variables first, constructors second, and methods third.
- Use a return type in methods. If nothing can/should be returned, use "void" signature

## Comments

**GOOD PRACTICE:**

- Add class description so that others can understand the purpose of the code in it:

```
/**
 * Prepare the world for the start of the program.
 * That is: create the initial objects and add them to the world.
 */
private void prepare() {}
```

- Include comments in the code if you suspect there are ambiguities or special circumstances surrounding it

- If you are too unsure what the code does, or whether it is a good solution, it might be a good idea to add reference the chapter or code line where you saw it before

## Variables

**ALWAYS**:

- Choose a name *starting with lowercase*

```
int playerScore = 0;
```

**NEVER**:

- Use numbers as initial character

```
~~int 4playerSetup = "A game of four";~~
```

- Reuse variable names for other purposes within classes:

```
int times = 5;
public void update() {
   myGame.setIntermediates(times);
   for (~~time~~ = 0; ~~time~~ < 10; i++ ) {
```

**GOOD PRACTICE:**

- When hard-coding(*) a value, you can describe it either by:

a. Creating a new variable for it

```
int maxParticipants = 4;
int[] playerScores = new int[maxParticipants];
```

b. Or, by placing a comment smoothly nearby:

```
int[] playerScores = new int[4]; // maximum number of participants
```

- Variables can be named after their type, shape, content, role or the combination of them:
    - `myInteger` (type)
    - `myArray` (shape)
    - `myChoice` (content)
    - `myMiddle` (role)
    - `myMiddleChoiceIntArray`
- Long and specific names are good for avoiding mistakes and remembering its use on later occasion, but bad for it takes longer to type and to read by others and yourself.

# Programmer's Vocabulary

### Hard-coded value 🔩

A number, string or other setting that is put into the code directly where it is used. The opposite is a *value derived from data* by the program. When a hard-coded value is seen by other programmer, it may appear to be taken from out of the blue, if not properly explained in a comment next to it.

GOOD PRACTICE: Gather the hard coded values in one block and annotate/comment "Initial values".

### Refactoring ✨

An umbrella term for different kinds of cleaning and restructuring of code. It ranges from plain variable renaming to building systems for object creation. For the record, the name "refactor" originates from mathematics and factorization as a way of cleaning up a solution.

GOOD PRACTICE: As soon as your code runs without errors, don't get too eager to jump into next task. Refactoring can be a breeze when done in small episodes along the way.

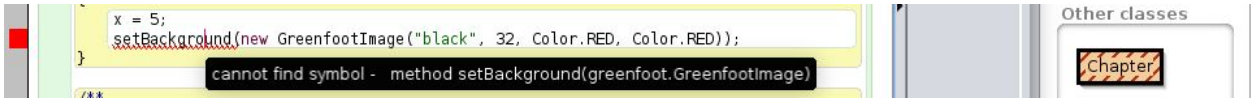
### Rubber-ducking 🦆

When the programmer discovers a solution hirself, while explaining the problem to a third person, or a rubber duck.

# Understanding common Error Messages

## Compilation errors

- Spelling, formatting, etc
- In Greenfoot, these error messages are shown directly in code editor.
    - *The Errors* are *_underlined_* after hitting `Ctrl-S` or clicking “Compile”
    - *The Error Messages* display when hovering mouse over underlined statement:




### "... Expected"

- mostly a missing } or ) or anything. Note that the message "...Expected" does not necessarily point out exactly where the error was committed, but it can give a clue! Eg. the indicated line may be in the same {}-code block as the error or just the line after. Sometimes the spelling and formatting is correct, but put in the wrong place, like a grammatical error! Also **"Illegal Start of an Expression"** is a kind of grammatical error.

### "Unclosed String Literal"

When a string is initiated in the code, the double quotes ("") must always contain the string and there should not be any line-breaks (when you pressed enter). Instead use \n inside the string to create a line-break.

### "Cannot Find Symbol"

A very common error, because it is quite general. When a variable name, class name or method name is referred to, but there is nothing to refer to, you will get this error. Possible mistakes:

- Forgetting to declare variable first
- Looking for something that was declared in wrong scope, or with wrong identifier (private, public, protected or without identifier)
- A class that is located in another file, but was not imported

### "Missing Return Statement"

Happens when there is a missing or misplaced “return” inside a method that is declared with a return type.

### "Missing Return Value"

When returning null or any data with type that doesn’t match method return type.

### "Cannot Return a Value From Method Whose Result Type Is Void"

### "Unreachable Statement"

For example if the return statement is not the final one inside a method. In general terms, anything that is always ignored by any strict logics.

### *"Variable <X> Might Not Have Been Initialized"*

Always assign something to a variable before trying to use it.

### *“(array) <X> Not Initialized”*

# Object orientation related errors

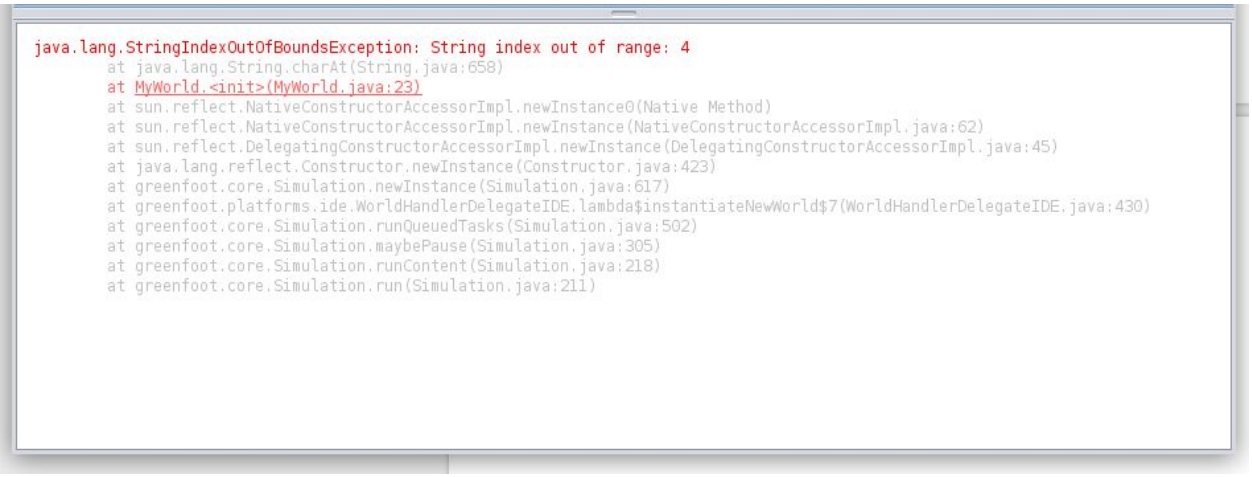
### *"Non-static method <X> cannot be referenced from a static context"*

- Non-static methods are always accessed using dot, on an instance variable:
  - `myString.charAt(4);//returns fourth character`
- Non-static instance-variables created with “new”-keyword
- Non-static variables always start with lowercase character

- Static variables are always declared with a “static” keyword
- Static context always start with uppercase character :
  - `String.valueOf(true); //returns "true"`
  - So the context is the String-class, where “valueOf” is a static method

# Runtime errors

- When a game is played, it can end in many ways. A *runtime error* is the kind of end where the game crashed!
- Testing a game in different manners will make these errors appear suddenly, but they may also happen as you make changes in game configuration, image files and *hard-coded values. (*what's a hard-coded value?)
- In greenfoot a Runtime Error pops up in a separate window as they happen:




Other common Error names:

NullPointerException
- "Computer can't treat nothing like there was something..."

IllegalArgumentException
- "Computer transforms input into output, but not just any input will do!"

NoSuchMethodException
- "Computers do not spell correct labels for their methods"

ArrayIndexOutOfBoundsException
- "No way I can go to a street address with number higher expected after the end of the street."