Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Pooja Rani

Solution
Assignment 05 — 14.10.2020 – v1.0
Moldable Development

Please submit this exercise by email to pascal.gadient@inf.unibe.ch before 21 October 2020, 10:15am.

Exercise 1 - General questions (2 pts)

i) Is code reading a problem? Justify your answer. **Answer:**

Yes, the task "code reading" represents a serious problem. According to various surveys, performed in a plethora of different companies, developers spend on average around 50+% of their work time on code reading. Developers specifically exposed to 3rd party code even invest up to 90+% of their time in understanding other's source code. Code reading is not just about reading the code, it's rather about understanding the implementation, the environment, and the architecture of a software project. Because the level of complexity in modern technology increases continuously, it's only a logical consequence that people spend significantly more time on understanding software.
That's why it is tremendously important to reduce the time required to gather the desired knowledge about a system. In an optimal environment the system should be very tangible; easy to inspect, as well as easy to access and interact with.

ii) Give an example (does not have to be from software) where a custom tool improved productivity by addressing a problem. Which tool did you choose, what is the addressed problem, and how did the tool improve productivity? **Answer:**

There exist many tools that help developers increasing the productivity:

- *Status indicators: Status indicators exist in different flavors increasing the awareness of a project or system state. In other words, they raise the overall focus on a specific aspect of a product. This increases the productivity, since real threats can be understood and tackled immediately. These indicators can be implemented in custom software that notify developers of finished or failed build processes, as audible notifications by playing a custom jingle in case of application errors, or as hard-plastic* Maneki-neko *implementation that resides near the entrance reminding people passing by of the current state of a project.*
- *IDE support: Tools in IDEs, often referred to as plug-ins, support developers while writing code. They offer diverse help: some of them support developers in writing secure code, while others check for spelling errors, or introduce conformance checks mandatory to pass for a successful software release. They are usually heavily customizable.*
- *Time scheduling: Traditional (offline) time tables and custom online calendars ease proper time planning as they are key factors with respect to productivity. With a tight schedule that is hardly to overcome, the stress level of employees increases and as a result the productivity will decrease substantially in the long-term. Hence, this customized tool and its correct use provide much value to any project.*
- *DIY tools: There exist numerous different tools that greatly increase the productivity for any mechanic: custom hammers, saws, screwdrivers, levers,* etc.

Exercise 2 - Inspector extensions (4 pts)

i) The GT inspector displays views from methods that contain the pragma <gtView>. How many classes in Pharo can visualize themselves, because they contain at least one method with that pragma? Provide your implementation. **Answer:**

There exist 547 classes with support for GT inspector visualizations.

```
results := Set new.
#gtView gtPragmas do:  [ :method |
  results add:  (method classBinding value name)].
results sorted:  [:a :b | a < b].
```

ii) Improve the `DateAndTime` class so that the GT inspector can visualize the date and time of such objects within a new view called "Human Readable".

The view must use the following format: `YYYY-MM-DD HH:MM` **Answer:**

```
DateAndTime>>gtHumanReadableFor:  aView
  <gtView>
  ^ aView textEditor
      title:  'Human Readable';
      text:  [self asStringYMDHM]
```

Please continue reading on the next page.

Exercise 3 - Live documents (4 pts)

i) What are the supported annotation names in live documents? In other words, which annotation names can you use in your live document code?

NB: Annotation names prefix live document code snippets. For example, `${class:Object}$` contains the annotation name `class` which tells the live document to use the appropriate visualization for classes. **Answer:**

You can find hints to all annotation name implementations in the class `GtDocumentConstants`. Each method name that ends with `AnnotationName` describes an annotation name. Currently, there are 11 annotation names supported: `changes`, `class`, `example`, `examples`, `explanation`, `icebergFile`, `inputFile`, `method`, `parametrizedExample`, `slides`, and `xdocList`.

```
annotationNameSelectors := OrderedCollection new.
allSelectors := GtDocumentConstants class selectors.
allSelectors do: [:each | (each endsWith: 'AnnotationName')
  ifTrue: [annotationNameSelectors add:
    (each gtRemoveSuffix: 'AnnotationName') ]].
annotationNameSelectors sort: [:a :b | a < b].
```

ii) Create a live document that always shows the current number of classes available in Pharo. You have to provide the live document code *and* its implementation. Your live document should look like this:

I consist of `18605` classes.

Step 1:
Create a method (using the correct pragma) that returns all classes. You are allowed to augment existing classes in GT.

Step 2:
Reference the method in your live document code. **Answer:**

Live document code:
```
I consist of ${example:BaselineOfGToolkit>>#allClasses|label=#size}$
classes.
```

Code in `BaselineOfGToolkit>>#allClasses:`
```
allClasses
<gtExample>
^ SystemNavigation default allClasses
```