Eclipse Advanced Übungen

Übung 1.1 – Working Sets

Erzeugen Sie verschiedene Working Sets, um verschiedene Sichten auf die Projekte zu erzeugen.

1.

Definieren Sie einen Working Set, der nur die Projekte mit dem Namen „Interessantes Projekt X" beinhalten und wählen Sie das Working Set aus. Es sollten nur noch die Projekte mit dem Namen „Interessantes Projekt X" angezeigt werden.
Kehren Sie danach zur ursprünglichen Ansicht zurück.

2.

Definieren Sie einen Working Set, der nur die service.xml-Dateien aus den „Interessantes Projekt X"-Projekten beinhaltet und wählen Sie das Working-Set aus. Es sollten nur die service.xml-Dateien angezeigt werden
Anmerkung : Dieses Beispiel ist sicherlich nicht praxis-tauglich, jedoch sind Filter nach bestimmten Dateien oft hilfreich (XLS der Anwendung, Konfigurationseinstellungen, ...)

3.

Erzeugen Sie in der Klasse „TestDebugging" im Projekt „Interessantes Projekt" drei Breakpoints: Zeile 20 (vor der Schleife), Zeile 24 (in der Schleife) und in Zeile 28 (nach der Schleife).
Erstellen Sie zwei Breakpoint Working Sets. Im ersten Working Set sollen die Breakpoints außerhalb der Schleife enthalten sein, im zweiten Working Set der Breakpoint innerhalb der Schleife.
Gruppieren Sie in der Breakpoint-View nach Working Sets und testen Sie das Aktivieren und Deaktivieren von Breakpoints auf Working Set Ebene.
Löschen Sie einen Breakpoint und ermitteln Sie, was mit dem Working Set passiert.

Übung 1.2 – Debugging

Implementieren Sie einen Detail Formatter für die Klasse „TestKlasse" im Projekt „Interessantes Projekt". Schauen Sie sich diese Klasse an und nehmen Sie an, dass es sich um eine Klasse handelt, in der „toString()" nicht implementiert werden kann.

1.

Konfigurieren Sie einen Detail Formatter in den Einstellungen „Java->Debug->Detail Formatter" für die TestKlasse, die folgende Ansicht im Debugger erlaubt:
Eigenschaft: *<eigenschaft der TestKlasse>*

NochEineTestklasse Eigenschaft: *<eigenschaft der NochEineTestKlasse, die von TestKlasse referenziert wird>*
NochEineTestklasse Eigenschaft 2: *<eigenschaft2 der NochEineTestKlasse, die von TestKlasse referenziert wird>*
Starten Sie die Klasse „TestDebugging“ und setzen Sie einen Breakpoint so innerhalb der Schleife, dass Sie sich die Variable „testKlasse“ im Debugger ansehen können.

2.

Deaktivieren Sie den Detail Formatter wieder und debuggen Sie erneut die Klasse „TestDebugging“. Vergleichen Sie die Ansicht für die Variable „testKlasse“.

Übung 1.3 Kata fizz-buzz

Erstelle ein Programm, welches die Zahlen von 1 bis 100 ausgibt. Dabei gilt:

1. Ist eine Zahl durch 3 teilbar, soll stattdessen „fizz“ ausgegeben werden
2. Ist eine Zahl durch 5 teilbar, soll stattdessen „buzz“ ausgegeben werden.
3. Ist eine Zahl durch 3 und 5 teilbar, soll „fizz buzz“ ausgegeben werden.

Zusatzanforderungen I

- Es soll ebenso „fizz“ ausgegeben werden, wenn eine Zahl die Ziffer „3“ enthält
- Es soll ebenso „buzz“ ausgegeben werden, wenn eine Zahl die Ziffer „5“ enthält

Zusatzanforderungen II

- Eine Zahl ist „super-fizz“, wenn sie durch 3 teilbar ist und die Ziffer 3 enthält. Danach werden die nächsten 3 „buzz“ unterdrückt
- Eine Zahl ist „super-buzz“, wenn sie durch fünf teilbar ist und die Ziffer 5 enthält. Danach werden das nächste „fizz“ unterdrückt

Übung 1.3 Kata Bank OCR

Es soll ein Programm geschrieben werden, welches eine Datei mit einer Reihe von schweizer Nummernkonten ausliest. Diese haben folgendes Format:

```
    _  _  _  _  _  _  _
|_||_||_| _| _|| || ||_   |
  ||_| _| _| _||_||_| _|  |


 _  _     _  _  _  _  _
 _| _||_||_||_ |_| _||_ |_|
|_ |_   | _||_||_| _||_|  |
```

…

Jede Zeile besteht aus 27 Zeichen, nach drei Zeilen erfolgt immer eine Leerzeile.

Schritt 1 – doc\OCR.txt

Gib die Nummernkonten der Reihe nach aus. In diesem Beispiel also:

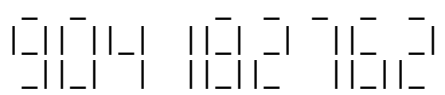
```
489330051
224968364
...
```

Schritt 2 – OCR-Checksum.txt

Die Nummernkonten bilden eine Prüfziffer. Die Kontonummer ist gültig, wenn für die folgende Summe Null ergibt. Die einzelnen Ziffern der Kontonummer sind dabei z_9 .. z_1;

$$c = \sum_{i=0}^{9} i\, z_i \; mod\; 11 = (9z_9 + 8z_8 + \cdots + z_1) \; mod\; 11 = 0$$

Zum Beispiel:



$$c = (9 * 9 + 8 * 0 + 7 * 4 + 6 * 1 + 5 * 8 + 4 * 2 + 3 * 7 + 2 * 6 + 2) \; mod\; 11$$
$$c = (81 + 0 + 28 + 6 + 40 + 8 + 21 + 12 + 2) \; mod\; 11$$
$$c = 198 \; mod\; 11$$
$$c = (18 * 11) \; mod\; 11 = 0$$

Prüfe für alle Nummernkonten auf eine korrekte Prüfziffer!

Schritt 3 – OCT-invalidRecognition.txt

Die Erkennung läuft leider nicht einwandfrei. Einige Zahlen werden nicht richtig erkannt. Protokolliere das Ergebnis, indem nach der erkannten Zahlenfolge der Status „ERR“ für eine fehlerhafte Prüfsumme und der Status „ILL“ für eine nicht gültige Ziffer angehangen wird. Nicht erkennbare Ziffern werden durch ein „?“ ersetzt. Also zum Beispiel:

```
457508000
664371495 ERR
86110??36 ILL
```

Schritt 4

Es stellt sich heraus, dass viele Fehler darauf beruhen, dass ein einzelner Strich nicht richtig erkannt wurde. z.B könnte ein „_“ statt einem „ “ erkannt worden sein, oder eine „ “ statt einem „|“. Eine 9 könnte also in Wahrheit eine 8 sein, wenn die Erkennung ein „|“ übersehen hat, oder eine 3 könnte eine 6 sein, oder eine 6 könnte auch eine 5 sein, wenn ein „|“ zu viel erkannt wurde.
Wenn es möglich ist, die Kontonummer durch den Austausch eines einzelnen Zeichens zu korrigieren, dann soll dies geschehen, und die Nummer erhält den Anhang COR. Sind mehrere Kontonummern möglich, erhält die Zeile den Zusatz AMB, mit einer Liste der möglichen Kontonummern. Falls der Status der Kontonummer immer noch unbekannt bleibt, ist der Zustand ILL.