Universität Zürich
Institut für Informatik

Software Engineering Übung 1

Programmverständnis, Dokumentation

1 Informationen

1.1 Daten

- Ausgabe Di 28.09.2010
- Abgabe So 10.10.2010 bis 23:59 Uhr
- Besprechung am Di 19.10.2010 um 12:15 Uhr

1.2 Formales

Die Lösungen sollen als PDF Datei mit dem Namen **Ex[*n*]_[*NameA_NameB_NameC*].pdf** abgegeben werden, wobei [*n*] die Nummer der Übung ist und [*NameA_NameB_NameC*] die Nachnamen der Gruppenmitglieder sind. Die PDF Datei sollte ausserdem ebenfalls Ihre Namen und Matrikelnummern beinhalten.

Mailen sie Ihre Lösungen vor dem Abgabetermin an `charrada@ifi.uzh.ch` und `wueest@ifi.uzh.ch`. Der Betreff der E-mail sollte mit **[SE EX HS10]** beginnen. Falls Sie zusätzliche Abgabematerialien (z.B. Source Code) haben, mailen Sie bitte ein Archiv (.zip-File), welches alle Dateien, einschliesslich dem PDF, enthält. Benennen sie das Archiv anhand der oben erwähnten Konventionen.

Die Übungen sollen in 3er Gruppen gelöst werden. Jedes Gruppenmitglied muss über alle Teile der Lösungen Auskunft geben können. Verspätete Abgaben werden korrigiert, aber nicht bewertet.

2 ZXing

2.1 Allgemein

Diese Aufgabe basiert auf der ZXing (Zebra Crossing) Bibliothek zum Verarbeiten von Strichcodes. Diese Java-Bibliothek kann mit der Kamera eines Mobiltelefones verwendet werden und unterstützt diverse 1D und 2D Strichcode-Formate. Das ZXing Projekt besteht aus verschiedenen Modulen, darunter die zentrale *core* Komponente zum Erzeugen und Entschlüsseln von Strichcodes, sowie diverse Client-spezifische Module für Google's Android, JavaME, JavaSE usw. Unser Fokus wird auf dem *core* Modul liegen, welches mehr als 26'000 Codezeilen (inklusive Kommentare und Leerzeilen) umfasst.

Zusätzliche Informationen zur ZXing Bibliothek finden Sie auf:
`http://code.google.com/p/zxing/`

Für diese Aufgabe verwenden wir zwei Komponenten:

- *Core*: Enthält die Hauptfunktionen zum Erzeugen und Entschlüsseln von Strichcodes. Sie werden den Quelltext dieses Moduls verwenden, um die Aufgaben zu lösen.
- *JavaSE Client*: Stellt ein simples GUI zur Verfügung. Sie können dieses GUI verwenden, um die Anwendung zu starten.

Damit alle die gleiche Ausgangslage haben, verwenden Sie bitte den auf unserer Website zu den Übungen angebotenen Quelltext.

2.2 Installation

In Eclipse, gehen Sie auf *File* und dann *Import...*. Im neuen Fenster klicken Sie auf *General* und dann *Existing Projects into Workspace*. Als Archiv wählen Sie die ZXing-core Datei. Klicken Sie auf *Finish*. Eclipse wird nun ein neues Projekt unter dem Namen *zxing-core* im Workspace anlegen. Auf die gleiche Weise importieren Sie *zxing-clientSE*. Da der Client das *core* Modul benötigt, müssen Sie dieses zum Build-Path hinzufügen. Dies erfolgt mit einem Rechtsklick auf das Client-Projekt, dann wählen Sie *Configure Build Path...* im Menü *Build Path*. Im *Projects* Tab, klicken Sie auf *Add...* und wählen Sie das *zxing-core* Projekt aus.
Um das GUI-Interface zu starten benützen Sie die Klasse *GUIRunner* aus dem Client-Projekt: Rechtsklick auf das *zxing-clientSE* Projekt, anschliessend wählen Sie *Run as* und *Java Application*. Als Main-Klasse bestimmen Sie *GUIRunner*.

Die Datei *"barcode.png"* (Quelle: Wikipedia) enthält einen UPC-A Strichcode, den Sie zum Testen des Programms verwenden können.

3 Verständnis des Quelltextes (12 Punkte)

3.1 Struktur (5 Punkte)

Für diese Aufgabe betrachten Sie folgende Klassen und Interfaces:

- Im Package `com.google.zxing`: `Reader`, `MultiFormatReader`
- Im Package `com.google.zxing.oned`: `OneDReader`, `CodaBarReader`, `MultiFormatUPCEANReader`, `UPCEANReader`, `UPCEReader`, `EAN8Reader`, `EANManufacturerOrgSupport`, `UPCEANExtensionSupport`.

Verwenden Sie ein Klassendiagramm, um die obengenannten Klassen und Interfaces, ihre Methoden sowie die Beziehungen untereinander darzustellen. *Private* und *protected* Methoden brauchen Sie nicht zu berücksichtigen.

3.2 Anwendungsbereich (2 Punkte)

Die Klassen zur Erzeugung und Entschlüsselung von Strichcodes implementieren die Interfaces `Writer` respektive `Reader` aus dem Package `com.google.zxing`.

Eruieren Sie, welche Arten von Strichcodes von diesem Programm unterstützt werden. Benennen Sie separat, welche Typen gelesen und welche geschrieben werden können.
Inwiefern unterscheidet sich die Klasse `MultipleFormatWriter` von den anderen Writer-Klassen?

3.3 Verhalten (5 Punkte)

Beschreiben Sie das Verhalten nach einem Aufruf der Methode *decode(BinaryBitmap image, Hashtable hints)* in der Klasse `com.google.zxing.qrcode.QRCodeReader`. Verwenden Sie dazu ein UML-Sequenzdiagramm und berücksichtigen Sie nur die folgenden Klassen:
`com.google.zxing.qrcode.QRCodeReader`,
`com.google.zxing.qrcode.decoder.Decoder`,
`com.google.zxing.qrcode.detector.Detector`,
`com.google.zxing.qrcode.decoder.BitMatrixParser`.

4 Verbessern des Quelltextes (8 Punkte)

4.1 Dokumentation (4 Punkte)

Die letzte Ziffer des Strichcodes ist die Prüfziffer. Sie berechnet sich aus den anderen Ziffern und wird zur Fehlererkennung verwendet. Erklären Sie anhand der Methode *UPCEANReader.checkStandardUPCEANChecksum(String s)* im Package `com.google.zxing.oned` wie die Prüfziffer für UPC und EAN Strichcodes berechnet wird. Ergänzen Sie die Dokumentation des Codes, indem Sie den Methodenkommentar für die Methode *checkStandardUPCEANChecksum(String s)* schreiben. Folgen Sie dabei den Richtlinien des Java Style Guide (erhältlich auf der Webseite der Übungen).

4.2 Verbessern des Codes (4 Punkte)

Wir möchten den *oneDReader* erweitern, damit er auch Strichcodes erkennt, die um 90 Grad gedreht wurden. Implementieren Sie die neue Funktionalität, und testen Sie sie anhand des Strichcodes in *"rotatedBarcode.png"*.
Tipp: Verwenden Sie die Methode *rotateCounterClockwise()* der Klasse `BinaryBitmap` (package `com.google.zxing`).