Gruppe: ueb11
Bearbeitet von: Andrea Müns
zuletzt bearbeitet: 04.06.2003

Dokumentationskonzept

Die Dokumentation ist integraler Bestandteil eines jeden Programms, und sollte eine Programmbeschreibung, Verwaltungsinformationen sowie die Quelltexte und deren Kommentierung enthalten.
Dabei besteht der Programmvorspann aus dem Name, der das Programm möglichst genau beschreibt, einer kurzgefassten Beschreibung des Programms einschließlich der Angabe, ob ein GUI-, ein Fachkonzept- oder ein Datenhaltungsprogramm bzw. eine entsprechende Klasse vorliegt, sowie Namen der Programmautoren und Versionsnummer mit Datum.

Die Designdokumentation soll aus folgenden Komponenten bestehen:

1. die Designbeschreibung
2. den mit javadoc extrahierbaren Funktionen
3. weiteren Blockkommentaren und der Inline- Kommentierung des Quelltextes
4. sowie dem Quelltext selbst
5. Anwender- sowie Installationsdokumentation

zu 1.)
Die Diagramme in der Designbeschreibung sollten nicht zu ausführlich erklärt werden. Hier reicht es aus, Attribute oder Methoden, deren Funktionalität nicht aus dem Namen allein ersichtlich ist, oder der Name nicht eindeutig ist, kurz zu erklären. Denn ein Neueinsteiger sollte getroffene Entscheidungen nachvollziehen können, ohne durch zu viele detaillierte Informationen den Überblick zu verlieren.

Zu 2.)
JavaDoc erstellt aus den strategischen Kommentarzeilen eine HTML- Datei, die Anwendern der Klasse einen Überblick über den Prototyp der Funktion gibt.

Bsp. Für eine gut kommentierte Klasse mit Einsatz von DocComments mit HTML- Tags:

```
/**
 * This class represents an Internet Protocol (IP) address.
 * <p>
 * Applications should use the methods <code>getLocalHost</code>,
 * <code>getByName</code>, or <code>getAllByName</code> to
 * create a new <code>InetAddress</code> instance.
 *
 * @author  Max Mustermann
 * @version 1.3, 02/23/02
 * @see     java.net.InetAddress#getAllByName(java.lang.String)
 * @see     java.net.InetAddress#getByName(java.lang.String)
 * @see     java.net.InetAddress#getLocalHost()
 * @since   JDK1.0
 */

public final
class InetAddress implements java.io.Serializable {
   ...
}
```

Zu 3.)
Durch gute Quelltextkommentierung kann eine leichte Einarbeitung bzw. Wiedereinarbeitung ermöglicht werden. Außerdem wird die Lesbarkeit des Programms verbessert, und Modifikation und Wartung erleichtert.

- Es sollten geeignete Kommentare gewählt werden. Zum Beispiel sollte im else- Teil jeder Auswahl die dort geltenden Bedingungen als Kommentar angegeben werden.
- Kurzkommentare, wie z.B. /* i wird um eins erhöht*/ sollten vermieden werden, und besser im Namen untergebracht werden
- wichtige Kommentare können zur Hervorhebung in einem Kommentarkasten untergebracht werden

Zu 4.)
Da die Implementierung bei unserem Projekt in Gruppenarbeit erfolgt, ist es hier noch wichtiger Richtlinien für guten Quellcode festzulegen. Diese Richtlinien sollen Hinweise geben, wie Programme korrekt zu schreiben und leicht zu warten sind. Dazu sollte der Quellcode folgende Kriterien erfüllen:

Gruppe: ueb11
Bearbeitet von: Andrea Müns
zuletzt bearbeitet: 04.06.2003
- konsistente Formatierung -> siehe Beispielcode
- leichte Wartbarkeit
- einfach zu lesen und zu verstehen
- frei von typischen Fehlern

Bezeichnerwahl:
Die Bezeichner (Namen) sollen die Funktion bzw. Aufgabe zum Ausdruck bringen. Dabei sollten keine problemfreien (besser problembezogene) oder technischen Bezeichner verwendet werden. Die Verwendung einzelner Buchstaben ist ungeeignet, da kurze Bezeichner nicht aussagekräftig sind, und es können sich leicht Tippfehler einschleichen. Ein erhöhter Schreibaufwand durch lange Bezeichner wird durch die Vorteile mehr als ausgeglichen. Durch benannte Konstanten kann bessere Lesbarkeit erzielt werden, und das das Programm ist änderungsfreundlicher.
Bsp.:
Der folgende Beispielcode soll eine konsistente Formatierung verdeutlichen:

```
/* Auto4.java */

public class Auto4
extends Auto2
implements Comparable
{
  public int compareTo(Object o)
  {
      int ret = 0;
      if (leistung < ((Auto4)o).leistung) {
        ret )= -1;
      } else if (leistung > ((Auto4)o).leistung) {
        ret = 1;
      }
      return ret;
  }
}
```

Hier ist zum Beispiel die Klammersetzung, das Einrücken zusammengehörender Strukturen sowie die einheitliche Form zu beachten. Am einfachsten lässt sich diese durch ein geeignetes Tool umsetzen, wie zum Beispiel TextPad unter Windows, mit dem dieses Beispiel erzeugt wurde.

Zu 5.)
Es soll ein Benutzerhandbuch erstellt werden, welches Informationen zur Installationsanleitung erhält. Außerdem ist dieses Benutzerhandbuch eine Zusammenfassung von vier einzelnen Handbüchern für die einzelnen Rollen: Übungsteilnehmer, Korrektor und Dozent. Diese können zur Veranschaulichung mit Screenshots unterlegt werden. Diese einzelnen Handbücher führen den Nutzer in der jeweiligen Rolle durch die Anwendung von der Registrierung angefangen. Im letzten Teil des Benutzerhandbuches kann evtl. auf vielleicht auftretende Probleme oder Hilfestellungen eingegangen werden.
Eine mögliche Gliederung für unser Benutzerhandbuch wäre also:

1. Produktbeschreibung
    1.1 Systemvoraussetzungen
2. Benutzerleitfaden
    2.1 Wissenswertes für den Administrator
    2.2 Rolle Dozent
    2.3 Rolle Korrektor
    2.4 Rolle Student
3. Problembehandlung

Eine Aufteilung der Benutzerleitfäden für die einzelnen Rollen in jeweils ein einzelnes Buch für die bestimmte Rolle ist nicht notwendig. Denn hier könnte man z.B. wichtige Hinweise o.ä. zur Handhabung des Software- Produktes mit in den Informationsteil des Programms online stellen, da dieses ja ein webbasiertes System ist. So hätte der Nutzer wichtige Hinweise sofort zur Hand, wenn evtl. Unklarheiten oder Probleme bzgl. der Bedienung auftreten.