Einschub: Aufgabe 12.1 (Breakpoint bei Assertion)

Problem: `assert` bricht ab
Lösung: Eigene Version, die nicht abbricht

Problem: Auf Makros kann man i.allg. keine Breakpoints setzen
Lösung: Mach einen Funktionsaufruf

```
#define Assert(c) if(!(c))\
  {cout << "\nAssertion violation "\
        << __FILE__ << ":" \
        << __LINE__ ;\
   assertionHandler();}

static void assertionHandler()
{ cout << endl; } // set breakpoint here
```

Problem: Danach ist man nicht mehr an der Stelle des Breakpoints
Lösung: `finish`

19.4 Unterschiede zu C++

19.4.1 Kein Operator Overloading

+ Vereinfacht Syntax
– Erschwert die Akzeptanz nützlicher Klassen (`complex`, LEDA `real`, dynamische/assoziative Arrays ...)

- Extrawurst für `String`. Philosophie von C++: Bias für Datentypen = Bias für Anwendungen

19.4.2 Ansatz zur Objektorientierung

- Klassenhierachie ist ein Baum mit Wurzel `Object` (In C++: Wald (Steppe? :-))
- Alle (ausser `static`) Member-Funktionen sind virtuell
- Fundamentale Datentypen sind ganz anders als Objekte (C++: annähnernde Gleichbehandlung). Aber: Es gibt korrespondierende Klassen (`int`$\leftrightarrow$ `Int`, ...)

19.4.3 Multiple Inheritance

C++ erlaubt einer Klasse von mehreren Basisklassen zu erben. “Multiple inheritance opens up a can of worms”.

Das erschwert Definition und Implementierung der Sprache.

Javas Antwort: Interfaces

Ein Interface ist:

- Eine virtuelle Klasse
- ohne **irgendwelche** Implementierungen und
- ohne Member-Variablen
- außer `static final` Members (Konstanten)

Eine Klasse darf von einer anderen Klasse (`extends`) und mehreren Interfaces (`implements`) erben. Ein Interface darf von beliebig vielen Interfaces erben.

19.4.4 Speicherverwaltung

Stapel: Lokale Variablen (fundamentale Typen und Objektreferenzen)

Heap: Fast alle Objekte und Arrays mit `new` alloziert

Global: Nur `static` Members

Kein `delete` nötig!

19.4.5 Garbage Collection:

Nicht mehr erreichbare Objekte werden **automatisch** gelöscht.

Das geht weil das Java-Laufzeitsystem vollständige Kontrolle über alle Referenzen hat. (C++'s Pointerarithmetik etc. macht das schwierig.)

+ Macht die meisten Destruktoren (`finalizer` in Java) überflüssig

+ Schließt viele Fehlerquellen aus

+ Macht den "reference only" Ansatz von Java erst praktikabel.

– Manchmal langsam (aber schneller als unausgegorene selbstgestrickte Speicherverwaltungen.)

– Problematisch bei Echtzeitanforderungen

19.4.6 Grund für Unterschiede?

Programmiererbild der Programmiersprachenerfinder (überspitzt):

C/C++: Gottgleiche Geschöpfe

Java: Ungezogene Kinder

Wenn das stimmt, ist der Erfolg beider Ansätze leicht zu erklären:

C/C++: Schmeichelt der Eitelkeit der Programmierer

Java: Programmierer **sind** (oft) wie ungezogene Kinder

19.5 Effizienz

19.5.1 Skalarprodukt

```
public class Sprod {
  static final int N = 16;
  static final int repeat = 10;
  static double a[] = new double[N];
  static double b[] = new double[N];

  static double scalarProduct0(
     double a[], double b[], int n)
  { double accu = 0;
    for (int i = 0;  i < n;  i++) {
      accu = accu + a[i] * b[i];
    }
    return accu;
  }

  public static void main(String argv[]) {
    double start, finish, result = 0.0;
    for (int i = 0;  i < N;  i++) {
      a[i] = i; b[i] = i/2.0;
    }
    start = System.currentTimeMillis();
    for (int j = 0;  j < repeat;  j++) {
      a[0] = j; result += scalarProduct0(a, b, N);
    }
    finish = System.currentTimeMillis();
    System.out.print("scalarProduct0 ");
    System.out.print(repeat * 2.0 * N / (finish - start) / 1e3);
    System.out.println("MFlop/s");
  }
}
```

≈ 19MFlops

19.5.2 Gehts noch langsamer?

Nehmen wir an wir wollen einen template-Code in C++, der Skalarprodukte für beliebige Datentypen berechnet, durch objektorientierten Java-Code ersetzen:

```
static double scalarProduct0(
   Double a[], Double b[], int n)
{ double accu = 0;
  for (int i = 0;  i < n;  i++) {
    accu = accu + a[i].doubleValue() *
                  b[i].doubleValue();
  }
  return accu;
}
```

≈ 4.1MFlops

19.5.3 Ist Java jemals schneller als C/C++?

- Vielleicht bei bestimmten Bibliotheken, die C nicht kennt
- Wenn man in C++ ungeschickte Speicherverwaltung (zuviel `new`) betreibt
- Man **kann** Java kompilieren (just in time Übersetzung ist aber nicht unbedingt die ultimate Antwort)

Aber das Fehlen von Templates und Wert-Objekten ist ein echtes Handicap.

19.6 Dokumentations-Kommentare

19.7 Abstract Windowing Toolkit

Ausgabe: Grafische Objekte wie Farben, Zeichensätze, Bilder, Punkte, Rechtecke, Polygone

Eingabe: Ereignisse

Fenster: Komponenten, Dialogboxen, Layoutmanager

etc. Druckerausgabe, Exceptions

19.7.1 Events

- Maustasten, Mausbewegungen
- Tastatur
- Fensterereignisse (Focus, schließen, . . .)
- . . .

Im einfachsten Fall wird automatisch entschieden

- welches Fenster (oder Knopf, . . .) für das Ereignis zuständig ist und
- was für ein Ereignis vorliegt

Für jeden Ereignistyp gibt es ein `EventListener` Interface. Der Benutzer muß also für alle zu unterstützenden Ereignisse eines Fensters (oder Button, . . .) ein Objekt angeben, daß diesen Listener-Typ implementiert.

19.8 Applets

„Anwendungchen“ das im „Sandkasten“ spielen darf. Kann in eine HTML-Seite einbezogen werden. Kein `main`. Statt dessen

init Datenstrukturen intialisieren

destroy Ganz zum Schluß. Warum kein `finalizer`?

start Sobald Applet sichbar wird

stop Wenn es unsichtbar wird (z.B. Scrollen) (wie beim Külschranklicht)

getAppletInfo,getParameterInfo Visitenkarte für den Browser

19.8.1 Beispiel: Hello World

```
<APPLET code="FirstApplet.class"
 width=150 height=100>
</APPLET>
```

19.8.2 Beispiel: Scribbling

19.9 Werkzeuge im Java Development Kit (JDK)

javac Compiler

java Interpreter

jdb Debugger. Kommandozeilenorientiert! Ähnlich gdb (Schusters Kinder tragen keine Schuhe?)

appletviewer Applets testen ohne Browser

javah Generiert `.h` und `.c` Dateien, die man vervollständigen muß um C-Code als *native method* aufrufen zu können.

jar Archivieren = zusammenpacken und komprimieren von Dateien. Compiler und Interpreter entpacken selbstständig

javadoc Generiere HTML-Dokumentation aus Quellen mit Doku-Kommentaren.

native2ascii Java-Quelltexte in länderspezifischer Codierung in für `javac` verdauliches ASCII verwandeln. (Für ISO-8859-1 (Latin-1) (z.B. Umlaute) nicht immer nötig)

19.10 Lokale und Anonyme Klassen

Lokale Klassen

Innerhalb einer member-function definierte Klasse.

Anonyme Klassen

Klassen ohne Namen? Ja!

- Nur ein Objekt
- Keine Konstruktoren
- Typischerweise Unterklasse einfacher Klassen.
- →Vor allem für Listener-Klassen des AWT

19.11 Anonyme Arrays

Beispiel

```
scalarProduct(new double[] {1.0, 1.0},
              new double[] {1.0,-1.0});
```


19.12 Reflexion

Sei `o` ein Objekt. Dann ist `o.getClass()` eine Instanz der Klasse `Class`, das die Struktur der Klasse beschreibt.

- Members
- Member-Functions
- ...

Wozu das?

Damit kann eine Methode einer Basisklasse herausfinden wie das aufrufende Objekt aussieht

19.13 Anwendung von Reflexion: Objektserialisierung

Gegeben ein (fast) beliebiges Objekt.
`someObjectOutputStream.writeObject(o)`
schreibt eine Repräsentation des Objekts.

`readObject()` liest Objekte ein.

Das funktioniert sogar mit komplizierten Graphdatenstrukturen!

Anwendungsbeispiele

- Speichern und Laden beliebiger Datenstrukturen
- Kommunikation zwischen Rechnern

verwandt: `clone()` macht Kopie. `equals()` macht Vergleich. (Was machen `=` und `==`?)

19.14 Java Beans

Klassen, die visuell mit einem Konstuktionswerkzeug bearbeitet werden können.

Ziel: Zusammenstellung einfacher Anwendungen ohne regelrechtes Programmieren.

- Reflexion erlaubt generische Editoren für member etc.
- Namenskonventionen: `get` ... , `set` ...
- Instanzen von `BeanInfo` geben nähere Information über bestimmte Beans
- Spezialisierte Editoren für bestimmte members. Beispiel: Mehrzeilige Inhalte eines member vom Typ `String`.
- Spezialisierte Editoren für Aspekte des ganzen Objekts
- Zumindest für AWT-Komponenten gibt es schon einiges

19.15 Internationalisierung

- Unicode Zeichen
- Datumsformate
- Ordnung von Zeichenketten
- Trennung von Text und Struktur (z.B. Menüs)

Ob das für Arabisch funktioniert?