2IP65-2IP70

Lecture 3

Code hygiene
Arrays
Assertions
Constants
Code hygiene

- code should be readable by humans
  (compiler takes care of readability by computer)
- you, your partner, your instructor, you next week, you next year
Rules and conventions

Variables
- use clear names
- use lowercase for variable names (uppercase for class names)
- add comment at declaration of variable: role in program, unity of measurement, etc.
- use one role per variable
Rules and conventions

- indent properly (NetBeans has Format function)
- use spaces and empty lines judiciously
- use comment to:
  - summarize and mark parts
  - explain role of variables
  - explain funny tricks (reconsider first!)
  - express assertions about the program state
Assertions

assert <boolean expression>;

- expression is evaluated,
  when true: go on
  when false: stop with message (throw exception)

- use to express property about current “program state” (values of variables)

- boolean expressions are limited, use comment and mathematics, English, etc.
class Abs {
    int x;
    int y;

    void calculateAbsolute {
        Scanner scanner = new Scanner( System.in );
        x = scanner.readInt();

        if (x>0) {
            assert x > 0;
            y = x;
            assert x > 0 && y == x;
            assert y == |x|;
        } else {
            assert x <= 0;
            y = -x;
            assert x <= 0 && y == -x;
            assert y == |x|;
        }
        assert y == |x|;
    }
}
Note

- assertion checking in Java is by default off
- switch it on: runtime option -ea
- project property in Netbeans
Invariant

- assertion that holds before a loop and after every iteration is called an invariant

- hence, after the loop hold:
  - invariant and negation of guard

  ```
  //assert I
  while ( G ) {
    //assert I ∧ G
    Body
    //assert I
  }
  //assert I ∧ ¬G
  ```

- Important means of analysis, documentation, and proof of loops

  ```
  //inv I
  while ( G ) {
    Body
  }
  //assert I ∧ ¬G
  ```

- usual notation:
Arrays ctd.

multidimensional arrays (matrices)

```java
int[][] matrix;
...
matrix = new int[5][7];

for (int x=0; x<4; x++) {
    for (int y=0; y<6; y++) {
        matrix[x][y] = x+y;
    }
}
```

Better:

```java
int[][] matrix;
...
matrix = new int[5][7];

for (int x=0; x<matrix.length; x++) {
    for (int y=0; y<matrix[x].length; y++) {
        matrix[x][y] = x+y;
    }
}
```
Arrays

- Index is int: you can do math!
- left neighbour of $a[i]$ is $a[i-1]$
- careful: expression $a[x]$ for $x<0$ or $x \geq a.length$ gives error (program crash)
Arrays

On input is a sequence of 100 grades (1-10);
Problem: which grade occurs most?
Highest grade – solution

count freqs

Scanner sc;
int[] frequencies;
int grade;
...
frequencies = new int[11];
// all elts of frequencies are 0
for (int i=0; i<100; i++) {
    grade = sc.nextInt();
    assert 0<grade && grade<11;
    frequencies[grade] += 1;
}

find highest

int hi;       // highest grade
int maxf = 0; // max freq
for (int g=1; g<11; g++) {
    if (frequencies[g]>maxf) {
        maxf=frequencies[g];
        hi = g;
    }
}
System.out.println(
    "highest grade: "+hi
);
Constants

- variables that don’t change after initialization
- syntax: `final <type> <name>;`
- examples:

```java
final int M;   // number of matches
...
System.out.println(“How many matches?”);
M = scanner.nextInt();
```

```java
final int N = 3;   // number of piles
...
```
Why use constants?

- documents function of variable
- protects against mistakes (or inconsistent change of plan)
- simplifies later changes: use named constants instead of literals
- useful in assertions: express state relating to previous state

```java
final int N;  // ...
//assert N >= 0;
... computation with f
//assert f = N!;
```