Date and Arrays

27 Jan 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Null reference

- To create an object, first declare it:
  - **Student joe;**
- Then create a new instance:
  - **joe = new Student(“Joe Smith”);**
  - **joe.getName();**
- Before an object is assigned, it has value null
- When accepting objects as function parameters, check to ensure they are not null references:
  - **public void copy(Student other) {**
    - **if (other != null) { …**

Initializing object attributes

- Set default values for attributes in constructor:
  - **public class Student {**
    - **String name;**
    - **Date birthdate;**
    - **public Student() { name = “Joe”;**
      birthdate = new Date(); }
- Or initialize in declaration (only for non-objects):
  - **public class Student {**
    - **String name = “Joe”;**
    - **public Student() {**
      birthdate = new Date(); }

Date

- Get the current date and time:
  - **import java.util.Date;**
  - **Date now = new Date();**
  - Stores number of milliseconds since midnight 1Jan1970 UTC (the "epoch")
- Format it in current timezone for display:
  - **import java.text.SimpleDateFormat;**
  - **DateFormat fmt = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");**
  - **fmt.format(now);**

DateFormat

- The date is universal, same across the globe
- How it is formatted depends on local timezone
- SimpleDateFormat creates a DateFormat formatter object that can convert between the Date (universal) and a string (localized)
  - Date → String: **fmt.format(date);**
  - String→Date: **fmt.parse("27 Jan 2010 15:00")**
- More info: see JavaSE documentation: Date, DateFormat

Arrays in Java

- Aggregate (compound/container) data type
- All entries have same type
- Size of array is fixed when array is allocated
  - But need not be known at compile-time
  - Arrays can be dynamically created
- Location in memory is usually contiguous
- Index into array using integer indices from 0 up to (size of array)-1
  - Indexing out-of-bounds raises ArrayIndexOutOfBoundsException

Working with arrays

- Declaring arrays:
  - `int numApples[];` `// or: int[] numApples;`
- Allocate array in memory:
  - `numApples = new int[10];`
- Initializing array entries:
  - `numApples[3] = 15;`
- Size of array:
  - `numApples.length` `// returns 10`

Array initializers and constants

- Initialize an array on one line:
  - `int numApples[] = {5, 3, 12, 0, 3};`
- Declare constants using the keyword final:
  - `final int numApples[] = {5, 3, 12, 0, 3};`
  - `final float pi = 3.14159265358979323846264;`
    - Values cannot be changed (even by code in the same class)
    - Initial value must be given in-line with declaration

Call-by-value vs. call-by-reference

- In Java, primitives (int, float, boolean, etc.) are passed by value
- Objects (including arrays) are passed by reference

Multidimensional arrays

- The element type of an array can be any type, including objects, including other arrays:

```
int image[][];
image = new int[width][height];
for (int x=0; x<width; x++)
    for (int y=0; y<width; y++)
        image[x][y] += 10;
```

- Rows may be different lengths:

```
image = new int[width][];
for (int x=0; x<width; x++)
    image[x] = new int[x];        // triangular array
```

Iterating through arrays

- Iterate through an array with a for loop:

```
for (int idx=0; idx < array.length; idx++)
    sum += array[idx];
```

- Java has an enhancement to the for loop:

```
for (int elt : array)
    sum += elt;
```

- But note elt is a copy of each element:
  - Can't use this to modify array:

```
for (int elt : array)
    elt *= 2;                    // doesn't change array!
```