More on Java

Object-oriented programming
Outline

- Primitive data types and objects
- Object references, assignment and comparison
- Parameter passing & return value
- this references
- Static members
- package
- Composition
- Standard I/O
- Command-line parameters

Readings:

- *Java How to Program*, 3.5-8, 6.3-8, 8.3-13
Data types

- Java is a strongly typed language
  - Every variable must have a declared type
  - There are two kinds of data types
    1. Primitive data types
      - Variables are manipulated via variable names
        - int a = 5;
        - if (a == b)...
    2. References types
      - Refer to objects
      - Objects are manipulated via references
        - GradeBook myGradeBook = new GradeBook();
Primitive data types

- Java’s primitive types:
  - Numerical: byte, int, long, float, double
    - No unsigned
    - Same size in all platforms
  - Logical: boolean (true/false)
  - Characters: char

- Primitive data are NOT objects
  - `int count = 0;`
  - `if (count == 5) ...`

- There’re corresponding wrapper classes
  - `Integer, Float, ...`
  - `Integer count = new Integer(0);`
## Primitive data types

<table>
<thead>
<tr>
<th>Types</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Wrapper types</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>16 bits</td>
<td>0x0</td>
<td>0xffff</td>
<td>Character</td>
</tr>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128 (-2^7)</td>
<td>+127 (2^7-1)</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32768 (-2^15)</td>
<td>32767 (2^15-1)</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2^31</td>
<td>+2^31 – 1</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-2^63</td>
<td>+2^63 - 1</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>IEEE754</td>
<td>IEEE754</td>
<td>Double</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>_</td>
<td>_</td>
<td>Boolean</td>
</tr>
<tr>
<td>void</td>
<td></td>
<td>_</td>
<td>_</td>
<td>Void</td>
</tr>
</tbody>
</table>
Where storage lives?

Where are primitive data, references and objects stored?

- Registers: Programmers can’t see or control
- Stack: Object references
- Heap: Objects
- Static storage: data that is available for the entire time a program is running
- Constant storage: Constant values are often placed directly in the program code
- Non-RAM storage: streams…
Where storage lives?

- In memory
  - code
  - static data
  - constants
  - temporary data
  - dynamic data

- static memory
- stack memory
- heap memory
Objects and Object references

class Date
{
  public:
    void setDate(...);
    int getDay();
    ...
  private:
    int day;
    int month;
    int year;
};

Date today = new Date();
today.setDate(2, 2, 2010);
Object references

- Object references store object locations in computer’s memory

- Objects are manipulated via references
  - Pointers to objects
  - Directly handle attributes and methods
  - No pointer operators
  - Assignments (=) of references do NOT copy object’s content

- Most references stay in stack or static memory like C/C++ pointers
Assignment operator “=”

- Copy the reference's content, NOT the referred object

```java
Integer m = new Integer(10);
Integer n = new Integer(20);

m = n;
n.setValue(50);
System.out.print(m);
```

50
Equality operators: “==“ and “!=“

- Compare content of the variables
  - Value of primitive data
  - Value of references
    - i.e. check if they point to the same object
    - NOT whether the content of the objects are the same

```java
Integer m1 = new Integer(10);
Integer m2 = new Integer(10);
System.out.println(m1 == m2);

int n1 = 1;
int n2 = 1;
System.out.println(n1 == n2);
```

false
true
Comparisons of object content

- **Method “equals”**
  - Pre-defined classes:
    - Ready to use
    ```java
    Integer m1 = new Integer(10);
    Integer m2 = new Integer(10);
    System.out.print(m1.equals(m2));
    ```
  - User-created classes:
    - `equals()` must be defined, otherwise it always returns `false`
    ```java
class MyInteger {
    private int value;
    public boolean equals(MyInteger other) {
        return (value == other.value);
    }
    ...
}
... MyInteger m1 = new MyInteger(10);
MyInteger m2 = new MyInteger(10);
System.out.print(m1.equals(m2));
```
Parameter passing & return value

- Pass-by-value
  - Argument’s content are copied to the stack
  - The **ONLY** mechanism allowed in Java
    - Java does NOT support pass-by-reference

- Two kinds of parameters:
  - Primitive types
    - parameter’s value is copied
    - parameters can be constants, e.g. 10, “abc”…
  - Object references
    - the reference's value (object location) is copied, NOT the referred object.
Parameter passing & return value

■ How pass-by-value works?

□ A parameter is in effect a local variable where the value of the corresponding argument is copied to.

Function definition

ReturnType foo(DataType v)
{
    //processing v
}

Function call

... foo(u); // u is of type DataType

 equivalence to

v is a local variable.

What happens when DataType is
• a primitive type?
• a reference type?

{(DataType v = u;
    //processing v
}

Parameter passing & return value

class Date {
    int year, month, day;
    public Date(int y, int m, int d) {
        year = y; month = m; day = d;
    }
    public void copyTo(Date d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    public Date copy() {
        return new Date(day, month, year);
    }
    ...
}

y, m, d are of primitive data type. They'll take the values of the passed parameters.

d is a reference. d will take the values of the passed parameter, which is an object location.

return a reference to the newly created Date object. Again, it's a value, not the object.
Parameter passing & return value

```java
... int thisYear = 2010;
Date d1 = new Date(thisYear, 9, 26);

class Date {
    int year, month, day;
    public Date(int y, int m, int d) {
        year = y; month = m; day = d;
    }
    public void copyTo(Date d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    public Date copy() {
        return new Date(day, month, year);
    }
    ...}
```
Parameter passing & return value

```java
class Date {
    int year, month, day;
    public Date(int y, int m, int d) {
        year = y; month = m; day = d;
    }
    public void copyTo(Date d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    public Date copy() {
        return new Date(day, month, year);
    }
    ... 
}
```

```java
Date d1 = new Date(thisYear, 9, 26);
Date d2 = new Date(2000, 1, 1);
d1.copyTo(d2);
```

```java
d = d2;
d.year = d1.year;
d.month = d1.month;
d.day = d1.day;
```
Parameter passing & return value

```java
class Date {
    int year, month, day;
    public Date(int y, int m, int d) {
        year = y; month = m; day = d;
    }
    public void copyTo(Date d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    public Date copy() {
        return new Date(day, month, year);
    }
    ...
}
```

```java
Date d2 = new Date(2000, 1, 1);
Date d3 = d2.copy();
Date temp =
    new Date(d2.day, d2.month, d2.year);
    d3 = temp;
```
Parameter passing

Remember!

- Pass-by-value ONLY
- Value of object references are NOT objects
Garbage collection

- To reclaim the memory occupied by objects that are no longer in use
- Programmers don’t have to disallocate objects
- Java Virtual Machine (JVM) performs automatic garbage collection
  - Method `finalize()` is called by JVM, not programmers.
  - Guarantee no memory leaks
- However, there’s no guarantee when/whether an object is freed before the program terminates
  - Might not needed as memory is still available
  - Clean-up tasks must be done explicitly by other “clean-up” methods
The **this** reference

- **this**: the reference that points to the *current* object (from inside).

- **usage of this**:
  - explicit reference to object’s attributes and methods
  - parameter passing and return value
  - calling constructor from inside constructor
Explicit reference using \texttt{this}

class Date {
    int year, month, day;
    public Date(int year, int month, int day) {
        \texttt{this.year} = year;
        \texttt{this.month} = month;
        \texttt{this.day} = day;
    }
    public void copyTo(Date d) {
        d.year = year;
        d.month = month;
        d.day = day;
    }
    ...
}

this as a parameter

class Document {
    Viewer vi; //reference to the document’s viewer
    ...

    Document(Viewer v) {
        vi = v;
        ...
    }

    void display() {
        //ask the object’s viewer
        //...to display the current document
        vi.display(this);
    }

    ...
}

this as a return value

class Counter {
    private int c = 0;
    public Counter increase() {
        c++;
        return this;
    }
    public int getValue() {
        return c;
    }
}
...
Counter count = new Counter();
System.out.println(count.increase().increase().getValue());
Call constructors using this

A constructor can be called explicitly only from inside another constructor, and can be called only once

class Date {
    private int year, month, day;

    public Date(int y, int m, int d) { ... }

    // copy constructor
    Date(Date d) {
        this(d.year, d.month, d.day);
        System.out.println("copy constructor called");
    }

    ...
}
Static methods and attributes

- Instance variables
  - Each instance has its own copy
  - Syntax: non-static data members

- Instance methods
  - MUST be invoked on particular instance
  - Syntax: non-static methods

- Class variables
  - ONE copy shared by all instances
  - Syntax: static data members

- Class methods
  - Can be invoked directly from the class (using class name)
  - Syntax: static methods
  - e.g. Math.sqrt()
public class Account {
    static double interestRate = 0.4;
    private String owner;
    private double balance;
    static double getInterestRate() {
        return interestRate;
    }
    public double endMonthCalculate() {
        balance += balance * interestRate;
    }
}

//main function to test Account class
public static void main(String args[]) {
    System.out.println( Account.getInterestRate() );
    Account myAccount = new Account ("An", 300000);
    myAccount.endMonthCalculate();
    System.out.println("After one month: "+ myAccount.getBalance())
}
public class Dummy {
    // number of Dummy objects
    static int counter = 0;
    static int count() {
        return counter;
    }

    private String name;

    public Dummy(String name) {
        counter++;
        this.name = name;
    }

    // main function to test Dummy class
    public static void main(String args[]) {
        System.out.println(Dummy.count());
        Dummy d1 = new Dummy("First Dummy");
        System.out.println(d1.count());
        Dummy d2 = new Dummy("Second Dummy");
        System.out.println(d1.count());
    }
}
Static methods and attributes

- Static methods
  - Can’t access non-static attributes
  - Can’t call non-static methods

- Why?
Design pattern: Singleton

- Singleton: *Ensure a class has only ONE instance, and provide a global point of access to it.*

<table>
<thead>
<tr>
<th>Singleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>- instance: Singleton</td>
</tr>
<tr>
<td>- Singleton()</td>
</tr>
<tr>
<td>+ getInstance(): Singleton</td>
</tr>
</tbody>
</table>

```java
if (instance == null) {
    instance = new Singleton();
}
return instance;
```

- Uses
  - In place of global variables
  - In system resource management
    - Avoid conflicting accesses from concurrent processes

- *Exercises:* Implement a Singleton class with data and test it.
Package: the library unit

- Object classes are grouped into packages
- Why? To manage *namespaces*, to prevent clashes of names.
- How?
  - Without explicit declaration, a class is placed in the “default package”
  - Classes inside one source file belong to one package
- Package access level
  - Default access level (without explicit declaration as private or public)
  - Objects of classes in the same package can access non-private members of one another
  - Can only create (new) objects of public classes in other packages
Package: How to

//Hello.java:

class HelloMsg {
    void sayHello() {
        System.out.println(”Hello, world!”);
    }
}

public class Hello {
    public static void main(String[] args) {
        HelloMsg msg = new HelloMsg();
        msg.sayHello();
    }
}

How to place HelloMsg in a package?
Package: Declaration

- a package statement appears as the first non-comment in the file

```java
// HelloMsg.java
package hanv;

public class HelloMsg {
    public void sayHello() {
        System.out.println("Hello, world!");
    }
}
```

package declaration with package name. The rest of the file belongs to the same package

Declared as public so that they can be used outside package hanv
Package: Usage

- Two ways:

```java
//Hello.java
import hanv.HelloMsg;

public class Hello {
    public static void main(String[] args) {
        HelloMsg msg = new HelloMsg();
        msg.sayHello();
    }
}
```

**1. Use the `import` statement to make the name(s) in the package available, once for all**

```java
//Hello.java
public class Hello {
    public static void main(String[] args) {
        hanv.HelloMsg msg = new hanv.HelloMsg();
        msg.sayHello();
    }
}
```

**2. Give the fully qualified name at every call**
Compile and run

- Compile
  javac HelloMsg.java -d <class_root_dir>
  javac Hello.java

- Run
  java Hello
Package – make it simple

- Where to put source files?
  - C:\java root directory
  - C:\java\hanv classes in hanv package

- Compile: **stay at the root!**
  - C:\java\> javac hanv\HelloMsg.java
  - equivalent to javac hanv\HelloMsg.java -d .
  - or javac hanv\HelloMsg.java -d c:\java
  - C:\java\> javac Hello.java

- Run
  - C:\java\> java Hello
Composition

- Objects can contain other objects (of non-primitive types) as attributes
- References as attributes must be instantiated using `new` or assigned to an existing object.
Composition

```java
public class Person {
    private String name;
    private Date birthDate;

    public Person(String name, Date birthDate) {
        this.name = name;
        this.birthDate = birthDate;
    }

    public String toString() {
        return String.format( "%s: Birthday: %s", name, birthDate);
    }
}

Date d = new Date(31, 12, 1999);
Person p1 = new Person("Bob", d);

Person p2 = new Person("Alice", new Date(1, 1, 2000));
```

non-primitive data as attribute

The contained object must be created.
Input / output

- Readings: Java How to Program. Chapter 14
- file-processing
  - read and write data in memory, in files and over network connections
- Input / output stream
  - ordered data that is read from or written to a file
- A Java program opens a file / network connection by creating an object and associating a stream of bytes or characters with it.
Standard I/O

- Three stream objects automatically created when a Java program begins executing:
  - `System.out`: standard output stream object
    - normally enables a program to output data to the screen (console)
  - `System.err`: standard error stream object
    - normally enables a program to output error messages to the screen
  - `System.in`: standard input stream object
    - normally enables a program to input bytes from the keyboard

- All three can be redirected to be sent to or read from a different location, such as a file on disk
  1. Using methods `setIn()`, `setOut()`, `setErr()`
  2. At command line (input and output only):
     ```
     C:\> type input.dat | java AJavaProgram > output.dat
     ```
Standard output and error streams

- System.out and System.err can be used directly
  - System.out.println("Hello, world!");
  - System.err.println("Invalid day of month!");
Standard input

- **System.in**
  - An InputStream object
  - must be wrapped before use

- **Scanner**: wrapper that supports input of primitive types and character strings
  - `next()`: get the next word separated by white spaces
  - `next>Type()`: get the next data item of type `Type`
  - `hasNext()`, `hasNext>Type()`: check if there’re data left to be read
  - `Type` can be `Int`, `Line`, ..
Standard input. Example

```java
// import the wrapper class
import java.util.Scanner;
...
// create Scanner to get input from keyboard
Scanner input = new Scanner(System.in);

// read a word
String s = sc.next();

// read an integer
int i = sc.nextInt();

// read a series of big intergers
while (sc.hasNextLong()) {
    long aLong = sc.nextLong();
}
```
Input from a text file. Example

```java
import java.util.Scanner;
import java.io.FileInputStream; import java.io.IOException;
...
public static void main(String args[]) {
    try {
        // create Scanner to get input from a file stream
        Scanner sc = new Scanner(new FileInputStream("test.dat"));

        String s = sc.next(); // read a word
        int i = sc.nextInt(); // read an integer
        while (sc.hasNextLong()) { // read a series of big integers
            long aLong = sc.nextLong();
        }
    }
    catch(IOException e) {
        e.printStackTrace();
    }
    sc.close();
}
```

To deal with errors such as file-not-found

Open and close the text file

Import required classes

To deal with errors such as file-not-found
Write to a text file. Example

```java
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.IOException;
...
public static void main(String args[]) {
    int i = 1; long l = 10;
    try {
        // create a printwriter to write output to a file stream
        PrintWriter out = new PrintWriter(new FileWriter("test.data"));

        // write to file
        out.println("Hello "+ i + " "+ l);
        out.close();
    } catch(IOException e) {
        e.printStackTrace();
    }
}
...
Command-line parameters

//CmdLineParas.java:
public class CmdLineParas {
    public static void main(String[] args) {
        //display the parameter list
        for (int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }
}

C:\>java CmdLineParas hello world
hello
world