Java

Accessing system resources

Security Manager

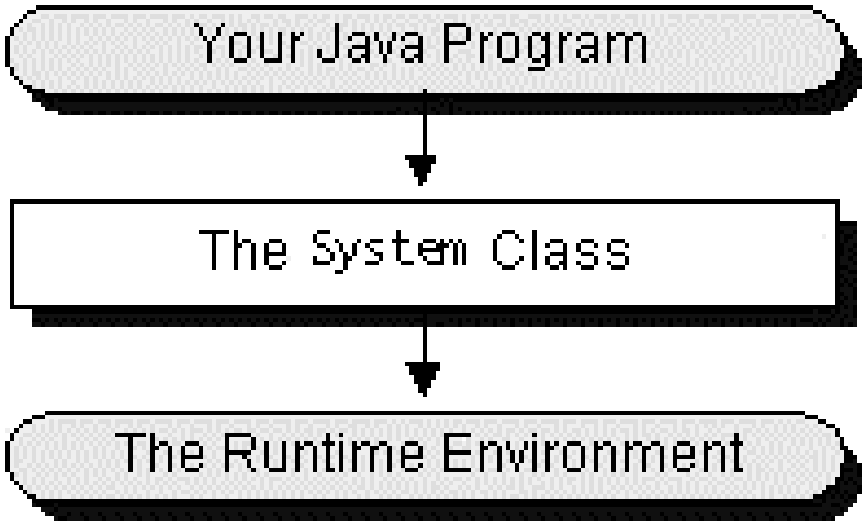
Presented by Bartosz Sakowicz

Accessing system resources

- Sometimes, a program requires access to system resources such as system properties, standard input and output, or the current time.
- Your program could make system calls directly to the window or operating system, but then your program would be able to run only in that particular environment. Each time you want to run the program in a new environment, you'd have to port your program by rewriting the system-dependent sections of code.
- The Java platform lets your program access system resources through a (relatively) system-independent API implemented by the System class and through a system-dependent API implemented by the Runtime class.

Accessing system resources(2)

The following diagram shows that the System class allows your Java programs to use system resources but insulates them from system-specific details:



System class

Unlike most other classes, you don't instantiate the System class to use it.

You cannot instantiate the System class - it's a final class and all of its constructors are private.

All of System's variables and methods are class variables and class methods - they are declared static.

To use a class variable, you use it directly from the name of the class:

System.out

You call class methods in a similar fashion:

System.getProperty(argument);

Standard I/O streams

The concept of standard input and output streams is a C library concept that has been assimilated into the Java environment. There are three standard streams, all of which are managed by the java.lang.System class:

Standard input--referenced by System.in

Used for program input, typically reads input entered by the user.

Standard output--referenced by System.out

Used for program output, typically displays information to the user.

Standard error--referenced by System.err

Used to display error messages to the user.

System properties

- The System class maintains a set of properties, key/value pairs, that define traits or attributes of the current working environment.
- When the runtime system first starts up, the system properties are initialized to contain information about the runtime environment, including information about the current user, the current version of the Java runtime, and even the character used to separate components of a filename.

System properties(2)

Key	Meaning
"file.separator"	File separator (for example, "/")
"java.class.path"	Java classpath
"java.class.version"	Java class version number
"java.home"	Java installation directory
"java.vendor"	Java vendor-specific string
"java.vendor.url"	Java vendor URL
"java.version"	Java version number
"line.separator"	Line separator

System properties(3)

Key	Meaning
"os.arch"	Operating system architecture
"os.name"	Operating system name
"os.version"	Operating system version
"path.separator"	Path separator (for example, ":")
"user.dir"	User's current working directory
"user.home"	User home directory
"user.name"	User account name

Reading system properties

The System class has two methods that you can use to read the system properties: getProperty and getProperties. There are different versions of getProperty:

System.getProperty("path.separator");

System.getProperty("subliminal.message", "Buy Java Now!");

You should use this version of getProperty if you don't want to risk a NullPointerException, or if you want to provide a default value for a property that doesn't have value or that cannot be found.

The getProperties method returns a Properties object that contains the complete set of system property key/value pairs.

Writing system properties

```
import java.io.FileInputStream; import java.util.Properties;
public class PropertiesTest {
    public static void main(String[] args) throws Exception {
        FileInputStream propFile =
        new FileInputStream( "myProperties.txt");
        Properties p = new Properties(System.getProperties());
        p.load(propFile);
        System.setProperties(p); // set the system properties
        System.getProperties().list(System.out);
    }
}
```

Forcing finalization

The Java runtime system performs memory management tasks for you. When your program has finished using an object, that is, when there are no more references to an object, the object is finalized and is then garbage collected. These tasks happen asynchronously in the background.

Before an object is garbage collected, the Java runtime system gives the object a chance to clean up after itself (finalization). You can force object finalization to occur by calling System's runFinalization method.

System.runFinalization();

This method calls the finalize methods on all objects that are waiting to be garbage collected.

Running the garbage collector

You can ask the garbage collector to run at any time by calling System's gc method:

System.gc();

You might want to run the garbage collector to ensure that it runs at the best time for your program rather than when it's most convenient for the runtime system to run it. For example, your program may wish to run the garbage collector right before it enters a compute or memory intensive section of code.

The amount of time that gc requires to complete varies depending on certain factors: How big your heap is and how fast your processor is, (e.g.). **Your program should only run the garbage collector when doing so will have no performance impact on your program.**

Security Manager

Each Java application can have its own security manager object that acts as a full-time security guard. The **SecurityManager** class in the java.lang package is an abstract class that provides the programming interface and partial implementation for all Java security managers.

By default an application does not have a security manager. That is, the Java runtime system does not automatically create a security manager for every Java application. **So by default an application allows all operations that are subject to security restrictions.**

To change this default behavior, an application must create and install its own security manager.

Security Manager(2)

You can get the current security manager for an application using the System class's getSecurityManager() method:

SecurityManager appsm = System.getSecurityManager();

getSecurityManager() returns null if there is no current security manager for the application so you should check to make sure that you have a valid object before calling any of its methods.

Writing a Security Manager

To write your own security manager, you must create a subclass of the SecurityManager class.

Your SecurityManager subclass overrides various methods from SecurityManager to customize the verifications and approvals needed in your Java application.

If the security manager approves the operation then the check*XXX*() method returns, otherwise check*XXX*() throws a **SecurityException.**

Security Manager example

- The policy implemented by example prompts the user for a password when the application attempts to open a file for reading or for writing. If the password is correct then the access is allowed.
- To impose a stricter policy on file system accesses, our example SecurityManager subclass must override SecurityManager's checkRead() and checkWrite() methods.
- SecurityManager provides three versions of checkRead() and two versions of checkWrite().

Security Manager example(2)

```
import java.io.*;
public class PasswordSecurityManager extends SecurityManager{
    private String password;
    private BufferedReader buffy;
    public PasswordSecurityManager(String p, BufferedReader b) {
        super();
        this.password = p;
        this.buffy = b;
     }
```

Security Manager example(3)

```
private boolean accessOK() {
    int c;
   DataInputStream dis = new DataInputStream(System.in);
    String response;
   System.out.println("What's the secret password?");
   try {
        response = dis.readLine();
        if (response.equals(password)) return true;
        else return false;
    } catch (IOException e){ return false; } }
```

Security Manager example(4)

```
public void checkRead(FileDescriptor filedescriptor) {

if (!accessOK()) throw new SecurityException("Not a Chance!"); }

 public void checkRead(String filename) {

if (!accessOK()) throw new SecurityException("No Way!"); }

public void checkRead(String filename, Object
     executionContext){

if (!accessOK()) throw new SecurityException("Forget It!"); }

public void checkWrite(FileDescriptor filedescriptor) {

if (!accessOK()) throw new SecurityException("Not!"); }

public void checkWrite(String filename) {

if (!accessOK()) throw new SecurityException("Not Even!"); }
```

Installing Security Manager

The main() method should begin by installing a new security manager:

```
try {
    System.setSecurityManager(new
    PasswordSecurityManager("MY PASSWORD"));
} catch (SecurityException se) {
    System.out.println("SecurityManager already set!");
}
```

You can set the security manager for your application only once. Any subsequent attempt to install a security manager within a Java application will result in a **SecurityException**.

Testing Security Manager

This is a simple test to verify that the PasswordSecurityManager has been properly installed:

```
try {

DataInputStream fis =

new DataInputStream( new
FileInputStream("inputtext.txt")); DataOutputStream fos =

new DataOutputStream( new
FileOutputStream("outputtext.txt"));

String inputString;

while ((inputString = fis.readLine()) != null) {
fos.writeBytes(inputString); fos.writeByte('\n'); } fis.close();
fos.close();

} catch (IOException ioe) { System.err.println("I/O failed"); }
```

Testing Security Manager(2)

Example of the output when you type in the password correctly the first time and incorrectly the second:

What's the secret password?

MY PASSWORD

What's the secret password?

Wrong password

java.lang.SecurityException: Not Even!

*at PasswordSecurityManager.**checkWrite**(PasswordSecurityManager.java:46)*

at java.io.FileOutputStream.(FileOutputStream.java)

at SecurityManagerTest.main(SecurityManagerTest.java:15)

Security Manager methods

In Security Manager class exists many methods possible to override. They are concerned with:

- sockets
- threads
- class loader
- file system
- system commands
- interpreter
- package
- properties
- networking
- windows