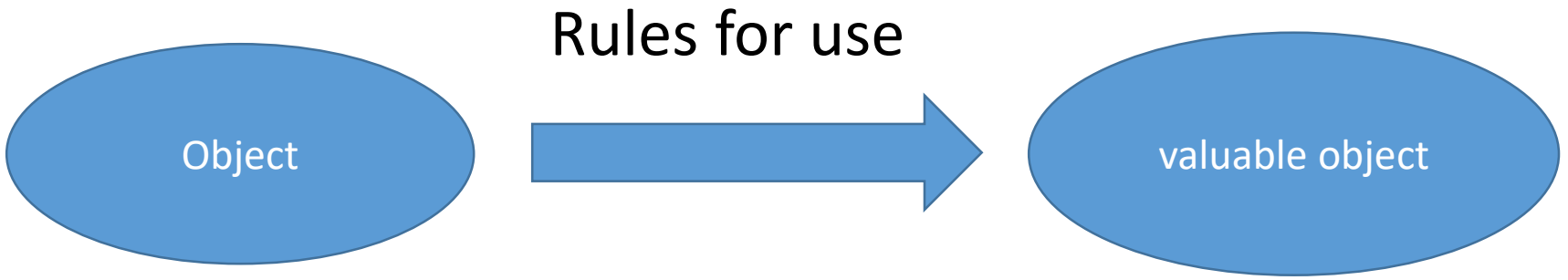
Object-oriented programming

Second semester

Lecture №10

Security

Security & OOP



Why your application cannot put the hard drive to sleep?

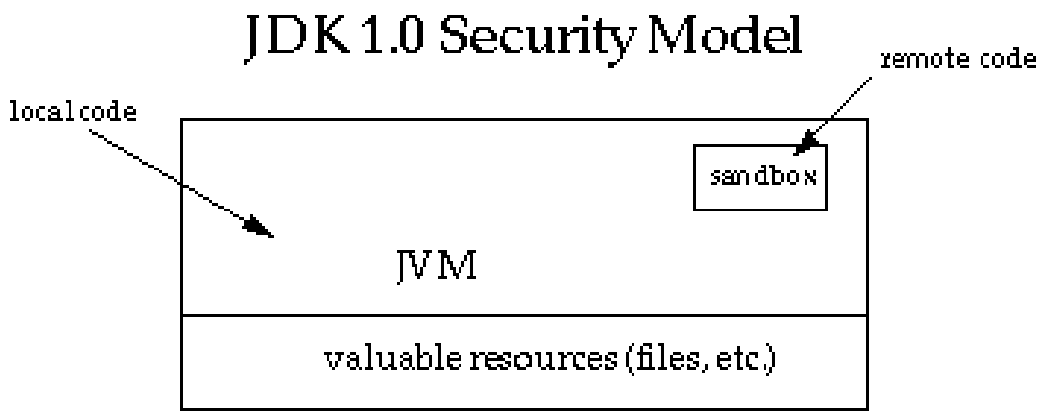
Example

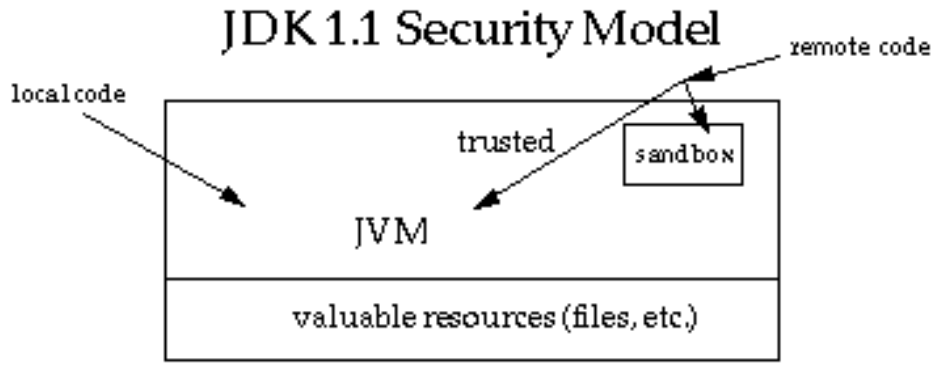
reader = new FileReader(“example.txt");

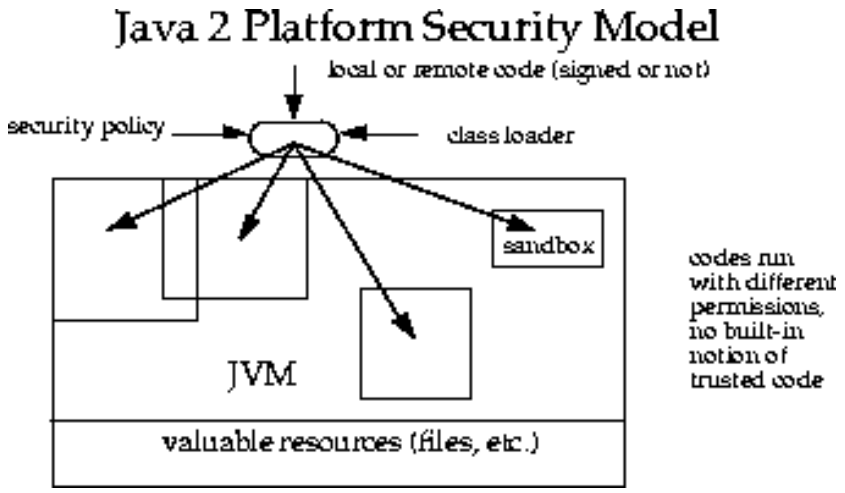
Exception in thread "AWT-EventQueue-1" java.security.AccessControlException:
access denied (java.io.FilePermission characteroutput.txt write)
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:323)
    at java.security.AccessController.checkPermission(AccessController.java:546)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:532)
    at java.lang.SecurityManager.checkWrite(SecurityManager.java:962)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:169)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:70)
    at java.io.FileWriter.<init>(FileWriter.java:46)

Note that the specific exception thrown in this case, java.security.AccessControlException, is a subclass of SecurityException

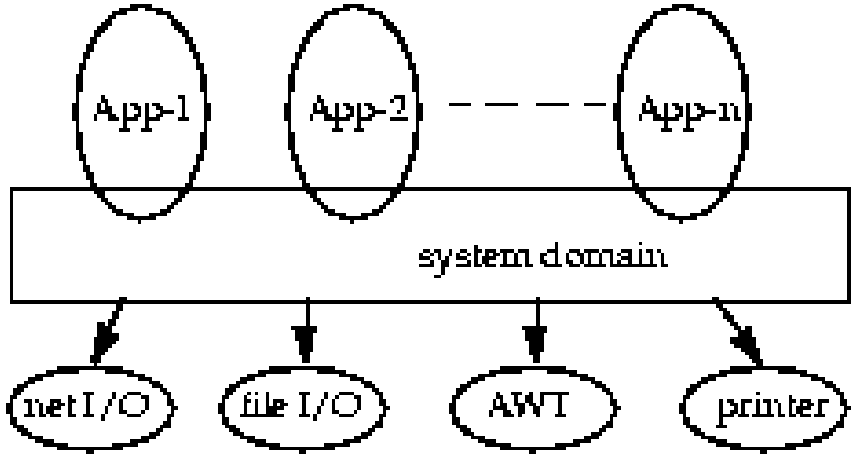
Security models

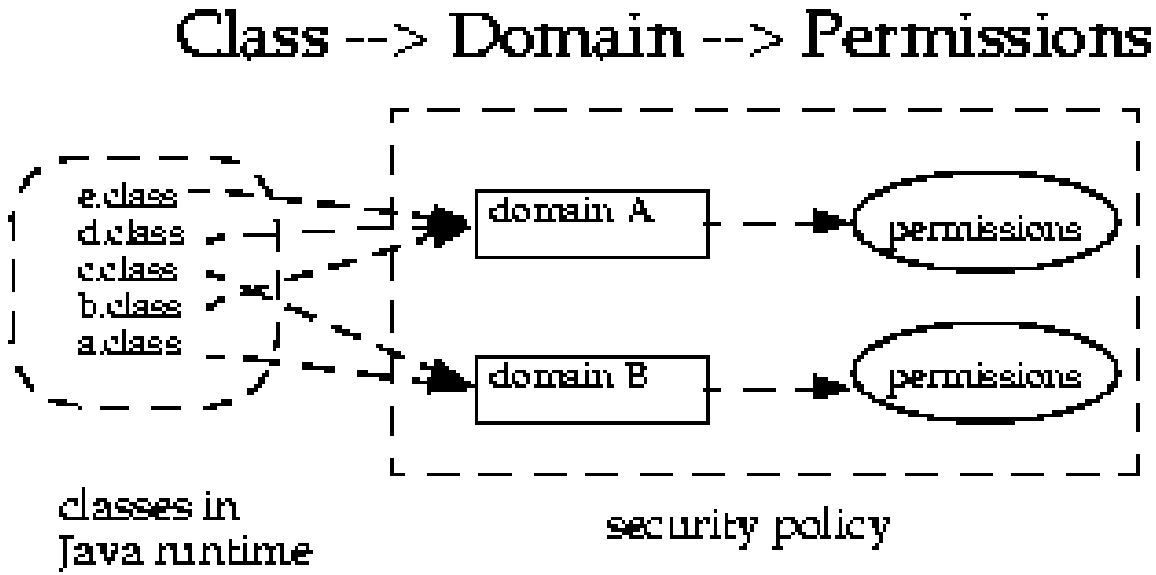






Protection Mechanisms

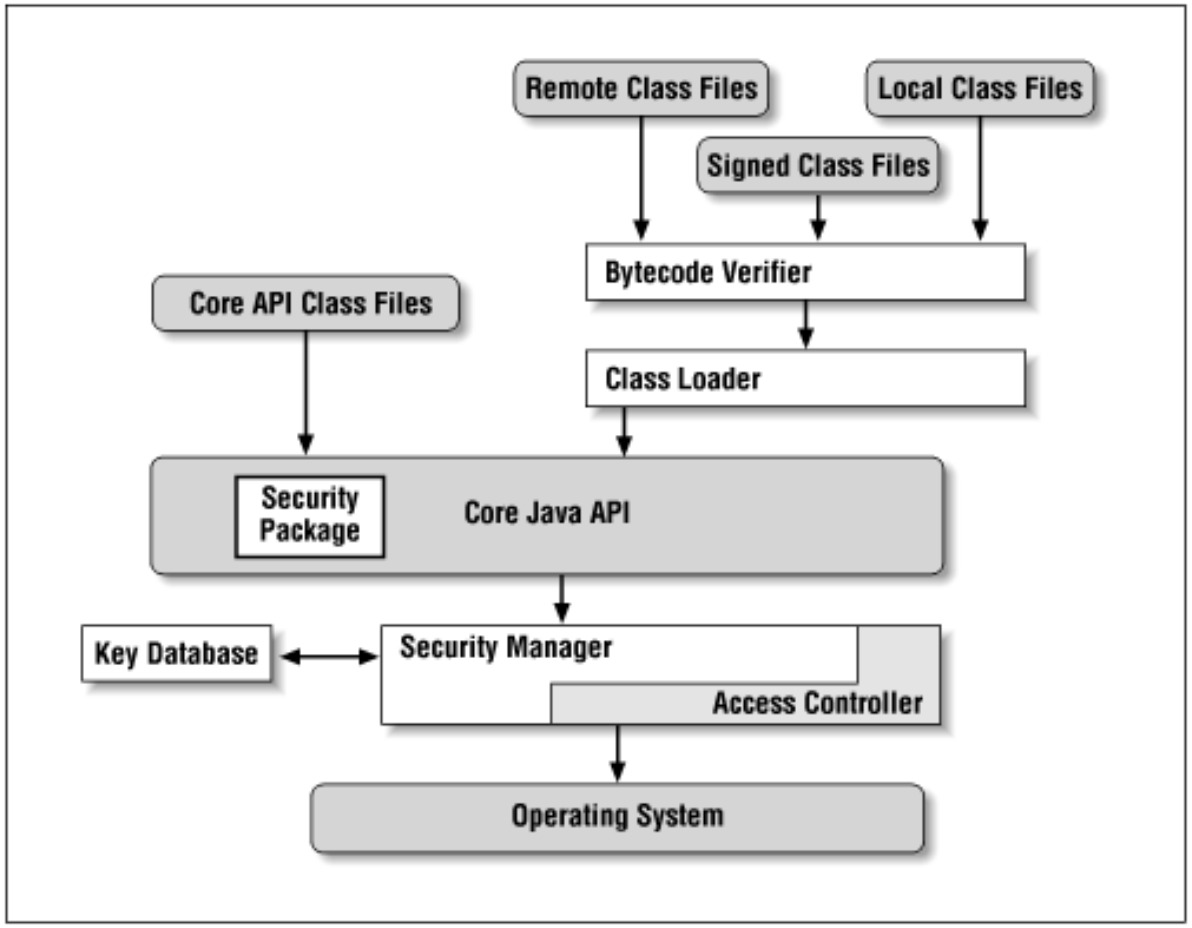




A fundamental concept and important building block of system security is the protection domain. A domain can be scoped by the set of objects that are currently directly accessible by a principal, where a principal is an entity in the computer system to which permissions (and as a result, accountability) are granted. The sandbox utilized in JDK 1.0 is one example of a protection domain with a fixed boundary.

Protection domains generally fall into two distinct categories: system domain and application domain. It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via system domains. The figure below illustrates the domain composition of a Java application environment.

Java Security




```
grant codeBase "http://www.example.com/*", signedBy "Li" {
    permission java.io.FilePermission "/tmp/*", "read";
    permission java.io.SocketPermission "*", "connect";
};
```

Domain rules

- Права доступа в контексте вызова какого-либо метода образуются пересечением прав доступа доменов безопасности, к которым принадлежат объекты образующие стек вызовов потока
- Таким образом, вызов метода объекта системного домена не добавляет прав вызвавшему его объекту из домена приложения
- Вызов метода объекта домена безопасности приложения из объекта системного домена (например «callback» для обработки событий) также не добавит прав объекту домена приложения
- Никакой домен не может «расширить» свои права вызвав, или будучи вызванным из домена с более расширенными правами
- Механизм вызова «doPriveleged» позволяет доверенному участку кода (обычно из системного домена) получить расширенные права доступа к ресурсам по отношению к правам доступа, действующим в контексте перед вызовом привилегированного кода

Example: FilePermission

The actions are: read, write, delete, and execute. Therefore, the following are valid code samples for creating file permissions:

```
import java.io.FilePermission;

FilePermission p = new FilePermission("myfile", "read,write");
FilePermission p = new FilePermission("/home/gong/", "read");
FilePermission p = new FilePermission("/tmp/mytmp", "read,delete");
FilePermission p = new FilePermission("/bin/*", "execute");
FilePermission p = new FilePermission("*", "read");
FilePermission p = new FilePermission("/-", "read,execute");
FilePermission p = new FilePermission("-", "read,execute");
FilePermission p = new FilePermission("<<ALL FILES>>", "read");
```