```
[...]
```

Utilizzare una libreria esterna come SerialKiller è molto utile per mettere le classi Java al riparo dalla vulnerabilità nota come "unsecure serialization". In pratica produce una sottoclasse "sicura " della classe usata da Java per la deserializzazione: ObjectInputStream.
Il codice Java, dopo aver importato tale libreria, viene modificato come segue:

Formato vulnerabile:

```
ObjectInputStream ois = new ObjectInputStream(is);
String msg = (String) ois.readObject();
```

Formato sicuro:

```
ObjectInputStream ois = new SerialKiller(is, "/etc/serialkiller.conf");
String msg = (String) ois.readObject();
```

7.2.10.10 Memorizzazione delle informazioni riservate

Non inserire all'interno del codice informazioni riservate, come chiavi crittografiche, passwords, certificati, etc. Informazioni personali non devono mai essere inserite in chiaro né devono essere presenti all'interno del codice o lasciati nella cache.

7.2.10.11 Packages

Creazione dei packages

I packages devono essere concepiti per organizzare in una forma logica le classi dell'applicazione; un package deve contenere funzionalità simili e omogenee.

Protezione dei packages

È necessario proteggere i package a livello globale, contro l'immissione di codice malevolo o alterato. Di seguito vengono riportati due esempi su come rispettare questa regola:
Esempio:

```
// Inserire la seguente linea nel file java.security properties.
// Ciò causerà un'eccezione nel loader defineClass non appena
// si proverà a definire una nuova classe all'interno del pacchetto,
// a meno che il codice non sia stato dotato del seguente permesso
// RuntimePermission("defineClassInPackage."+package)
[...]
package.definition=Pacchetto1 [,Paccchetto2,...,Pacchetton]
[...]
```

È anche possibile inserire le classi del pacchetto in un file jar. In questo modo nessun codice può ottenere il permesso ad ampliare il pacchetto e non c'è quindi motivo di modificare il file java.security properties.
È necessario proteggere l'accesso. Ciò può essere fatto inserendo la seguente linea nel file java.security properties:

```
[...]
package.access=Pacchetto 1 [,Pacchetto 2,...,Pacchetto n]
[...]
```

Ciò causerà un'eccezione nel loader loadClass non appena si proverà ad accedere ad una classe all'interno del pacchetto, a meno che il codice non sia stato dotato del seguente permesso:

```
[...]
RuntimePermission("accessClassInPackage."+package)
[...]
```

7.2.10.12 Gestione delle eccezioni

Tutti i null pointer devono essere gestiti, di modo che il programma sia robusto e non dia origine a "stack trace" incontrollati . La forma corretta nell'esempio che segue, mostra come utilizzare le capacità di logging di Java per mantenere traccia delle eccezioni.

Esempio:

Forma non corretta

```
import java.io.*;
import java.util.*;
public class BadEmptyCatch {
  List quarks = new ArrayList();
  quarks.add("hello word");
  FileOutputStream file = null;
  ObjectOutputStream output = null;
  try{
   file = new  FileOutputStream("quarks.ser");
output =new ObjectOutputStream(file);
   output.writeObject(quarks);
  }
  catch(Exception exception){System.err.println(exception);
  }
  finally{
    try {
    if (output != null) {
     output.close();
    }
   }
   catch(Exception exception){
    }
  }
```

Forma corretta:

```
import java.io.*;
import java.util.*;
import java.util.logging.*;

public class ExerciseSerializable {

 public static void main(String args) {
  List quarks = new ArrayList();
  quarks.add("hello word");
  ObjectOutput output = null;
  try{
   OutputStream file = new FileOutputStream( "quarks.ser");
   OutputStream buffer = new BufferedOutputStream( file );
output =
new ObjectOutputStream(buffer);  output.writeObject(quarks);
  }
  catch(IOException ex){
   fLogger.log(Level.SEVERE, "Cannot perform output.", ex);
  }
  finally{
   try {
    if (output != null) {
     output.close();
    }
   }
   catch (IOException ex ){
    fLogger.log(Level.SEVERE, "Cannot close output stream.", ex);
   }
  }
```

O si specifica la clausola throws e si rinvia quindi la cattura dell'errore alla classe chiamante, oppure lo si gestisce localmente. In questo caso bisogna evitare di raggruppare le eccezioni in un blocco di eccezioni generico, in quanto ciò rappresenterebbe una perdita di informazioni importanti.

Esempio:

Forma non corretta

```
import java.io.*;
```