```
}
```

Esempio 2:

Forma non corretta:

```
public void useDate(Date date) {
        if (isValid(date))
                scheduleTask(date);
}
```

Forma corretta:

```
public void useDate(Date date) {
    Date copied_date = new Date(date.getTime());
    if (isValid(copied_date))
           scheduleTask(copied_date);
}
```

7.2.10.5 Definizione delle classi

Evitare l'utilizzo di classi interne (inner classes). In casi del tutto eccezionali, comunque, le classi interne devono sempre essere definite come private.

Esempio:

Forma non corretta:

```
package esempio;
public class MyFirstClass {
  [...]
  private class MySecondClass {
  }
  [...]
}
```

Forma corretta:

```
package esempio;
public class MyFirstClass {
[...]
}
class MySecondClass {
[...]
}
```

Per tener conto dei rilasci, è opportuno inserire un codice di versione per ogni classe, collocandolo all'interno di una variabile pubblica final, ed effettuare i controlli per la coerenza di versione sulle classi del package.

7.2.10.6 Codice e permessi speciali

Le classi Java non dovrebbero effettuare operazioni di sistema diretti. Non dovrebbero cambiare i permessi sul file system, né aprire socket, né caricare librerie dinamiche attraverso la System.loadLibrary o la Runtime.getRuntime.loadlibrary, ecc.

Se una di queste operazioni dovessere rendersi necessaria, occorrerà documentarne le motivazioni e procedere con lo sviluppo nella massima sicurezza.

In generale, come già accennato, minori sono i privilegi, più è sicura l'applicazione.

7.2.10.7 Esecuzione dei comandi di sistema

Supponiamo che un aggressore assegni alla variabile filename un valore del tipo:

```
filename ="joe; /bin/rm -rf /*";
```

Nell'esempio sotto riportato verrà eseguito il codice malevolo (forma non corretta); nella forma corretta, il codice malevolo sarà, invece, ignorato.

Esempio:

Forma non corretta:

```
void method (String filename) {
        System.exec("more " + filename);
```