5. Deadlocks

Sie haben im Skript gesehen, dass sich Deadlocks verhindern lassen, indem Sie

- die Ressourcen (Locks) in eine Reihenfolge bringen
- und in allen Threads die Ressourcen immer in dieser festen Reihenfolge anfordern.

Das macht das gleichzeitige Auftreten von Code-Abschnitten der Form

```
lock (A);                lock (B);
lock (B);                lock (A);
...
unlock (B);              unlock (B);
unlock (A);              unlock (A);
```

unmöglich. Da die Beachtung dieser Regel (wir nennen Sie *Regel A*) umständlich ist, wird ein alternatives Verfahren (*Regel B*) vorgeschlagen: In der Anwendung wird ein zusätzliches Lock `super` definiert, das immer vor dem ersten Aufruf von `lock()` als erstes gelockt werden muss. Wenn `super` gelockt wurde, dürfen die übrigen Ressourcen in beliebiger Reihenfolge gelockt werden. Erst nachdem alle regulären Ressourcen zurück gegeben wurden, darf der Code auch `super` zurückgeben. Die beiden obigen Code-Blöcke sind nach den neuen Regeln also zulässig, erhalten aber die folgende Form (neuer Code **fett**):

```
lock (super);            lock (super);
lock (A);                lock (B);
lock (B);                lock (A);
...
unlock (B);              unlock (B);
unlock (A);              unlock (A);
unlock (super);          unlock (super);
```

a) Warum sorgt *Regel A* immer für Deadlock-Freiheit?

b) Sorgt auch *Regel B* immer für Deadlock-Freiheit?

c) Es ist viel einfacher, beim Programmieren *Regel B* umzusetzen, als *Regel A* umzusetzen. Aber ist dieser Ansatz empfehlenswert?