Kapselung und Methodenbindung: Javas Designprobleme und ihre Korrektur

Dipl.-Inform. Peter Müller
Prof. Arnd Poetzsch-Heffter
Fernuniversität Hagen

Gliederung

- Einführung: Methodenbindung und Kapselung
- Überschreiben paketlokaler Methoden
  - Korrigierte Methodenauswahl
- Überschreiben geschützter Methoden
  - Verfeinerung von Zugriffsrechten
- Zusammenfassung

Einführung

- Dynamische Bindung
  - Überschreiben von geerbten Methoden
  - Bindung zwischen Aufrufstelle und Implementierung zur Laufzeit in Abhängigkeit vom Typ des Zielobjektes
- Kapselung
  - Verbergen von Implementierungsdetails
  - Zugriffsmodi in Java:
    - privater Zugriff (private access)
    - paketlokaler Zugriff (default access)
    - geschützter Zugriff (protected access)
    - öffentlicher Zugriff (public access)

Wechselwirkungen

- Überschreiben von Methoden
  - Nur zugreifbare Methoden werden überschrieben (sonst Neudefinition)
- Zugreifbarkeit überschriebener Methoden
  - Überschreibende Methoden mindestens so zugreifbar wir überschriebene Methode
- Komplexes Zusammenspiel der Konzepte
- Ziel: Weitgehende Orthogonalisierung

Javas Methodenauswahlalgorithmus

```
class T {
   protected void m( ) { ... }
}

class S extends T {
   public void m( ) { ... }
}

class U extends S {
}


   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit
  3. Festlegung des Aufrufmodus (virtual/nonvirtual)

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit
  3. Festlegung des Aufrufmodus (virtual/nonvirtual)
- Dynamisch:
  4. Bestimmung des Zielobjektes

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit
  3. Festlegung des Aufrufmodus (virtual/nonvirtual)
- Dynamisch:
  4. Bestimmung des Zielobjektes
  5. Lokalisierung einer Methode mit passender Signatur, abhängig vom Typ des Zielobjekts

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Javas Methodenauswahlalgorithmus

- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit
  3. Festlegung des Aufrufmodus (virtual/nonvirtual)
- Dynamisch:
  4. Bestimmung des Zielobjektes
  5. Lokalisierung einer Methode mit passender Signatur, abhängig vom Typ des Zielobjekts

```
class T {
   protected void m( ) { ... }
}
class S extends T {
   public void m( ) { ... }
}
class U extends S {
}

   T v = new U( );
   v.m( );
```

Anforderungen

- Auswahl der statisch gefundenen Methode oder einer diese überschreibende
- Zugreifbarkeit der dynamisch ausgewählten Methode für den Aufrufer

Probleme paketlokaler Methoden

```
package PT;

public class T {
   void m( ) { ... }
}


   T v = new PS.S( );
   v.m( );
```

```
package PS;

public class S extends PT.T {
   public void m( ) { ... }
}
```

Probleme paketlokaler Methoden

```
package PT;

public class T {
   void m( ) { ... }
}


   T v = new PS.S( );
   v.m( );
```

```
package PS;

public class S extends PT.T {
   public void m( ) { ... }
}
```

- T:m und S:m sind verschiedene Methoden gleicher Signatur

Probleme paketlokaler Methoden

```
package PT;
public class T {
   void m( ) { ... }
}

   T v = new PS.S( );
   v.m( );
```

```
package PS;
public class S extends PT.T {
   public void m( ) { ... }
}
```

- T:m und S:m sind verschiedene Methoden gleicher Signatur
- Statische Deklaration ist T:m

Probleme paketlokaler Methoden

```
package PT;

public class T {
   void m( ) { ... }
}

   T v = new PS.S( );
   v.m( );
```

```
package PS;

public class S extends PT.T {
   public void m( ) { ... }
}
```

- T:m und S:m sind verschiedene Methoden gleicher Signatur
- Statische Deklaration ist T:m
- Dynamisch wird S:m ausgewählt

Korrigiertes Auswahlverfahren

- Dynamisch ausgewählte Methode muß statische Deklaration überschreiben
- Dadurch konzeptionell Vereinheitlichung von privaten und nicht privaten Methoden
- Höhere Orthogonalität
- Statisch:
  1. Bestimmung der statischen Deklaration
  2. Prüfung der Zugreifbarkeit
  3. entfällt
- Dynamisch:
  4. Bestimmung des Zielobjektes
  5. Lokalisierung einer Methode mit passender Signatur, die die statische Deklaration überschreibt

Probleme geschützter Methoden

```
package PT;
public class T {
   protected void m( ) { ... }
}

class C {
  public void foo( ) {
     T v = new PS.S( );
     v.m( );                  }
}
```

```
package PS;



public class S extends PT.T {
   protected void m( ) { ... }
}
```

Probleme geschützter Methoden

```
package PT;
public class T {
   protected void m( ) { ... }
}

class C {
  public void foo( ) {
     T v = new PS.S( );
     v.m( );                    }
}
```

```
package PS;


public class S extends PT.T {
   protected void m( ) { ... }
}
```

- S:m überschreibt T:m

Probleme geschützter Methoden

```
package PT;
public class T {
   protected void m( ) { ... }
}

class C {
  public void foo( ) {
     T v = new PS.S( );
     v.m( );                    }
}
```

```
package PS;


public class S extends PT.T {
   protected void m( ) { ... }
}
```

- S:m überschreibt T:m
- Statische Deklaration ist T:m, zugreifbar für C

Probleme geschützter Methoden

```
package PT;
public class T {
   protected void m( ) { ... }
}

class C {
  public void foo( ) {
     T v = new PS.S( );
     v.m( );                    }
}
```

```
package PS;


public class S extends PT.T {
   protected void m( ) { ... }
}
```

- S:m überschreibt T:m
- Statische Deklaration ist T:m, zugreifbar für C
- Dynamisch wird S:m ausgewählt, nicht zugreifbar für C

Überschreiben geschützter Methoden

- Überschreiben von geschützten Methoden in anderen Paketen nur mit öffentlichen Methoden

```
package PT;
public class T {
   protected void m( ) { ... }
}

class C {
  public void foo( ) {
     T v = new PS.S( );
     v.m( );                  }
}
```

```
package PS;


public class S extends PT.T {
   public void m( ) { ... }
}
```

- Aufweichen der Kapselung

Verfeinerte Zugriffsrechte

- Trennung des Zugriffs für Subklassen und Paket
- Wiedereinführung des Zugriffsmodus „privat geschützt“ (private protected)
  - Zugriff nur für Sub- und Superklassen

```
package PT;

public class T {
   private protected void m( ) { ... }
}
```

```
package PS;

public class S extends PT.T {
   private protected void m( ) { ... }
}
```

Zusammenfassung

- Komplexes Zusammenspiel von Kapselung und Methodenauswahl
  - Widersprüche in Sprachspezifikation
  - Fehler in Compilern
- Korrigiertes und vereinfachtes Auswahlverfahren
  - Keine Unterscheidung zw. virtual und nonvirtual
  - Höhere Orthogonalität
- Verfeinerte Zugriffsmodi
  - Überschreiben geschützter Methoden
  - Zusätzlicher Modus „privat geschützt“

Ausblick

- Arbeitsumfeld
  - DFG-Projekt „Lopex“
  - Spezifikation und Verifikation von Java Programmen
  - Erstellung einer Hoare-Logik für Java
- Weitere Informationen
  - www.informatik.fernuni-hagen.de/pi5/publications.html
  - Peter.Mueller@fernuni-hagen.de
  - Arnd.Poetzsch-Heffter@fernuni-hagen.de