

A thick black L-shaped frame is positioned on the left and bottom edges of the slide, framing the content.

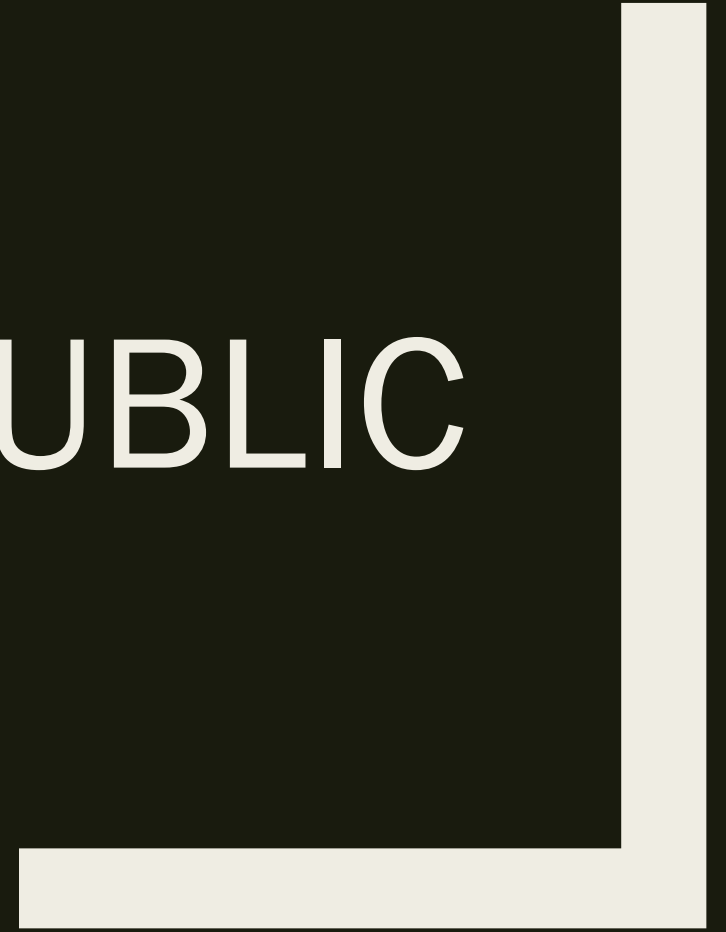
ACCESS MODIFIERS

Sichtbarkeitsmodifizierer

Sichtbarkeitsmodifizierer

- Sichtbarkeitsmodifizierer
- Sichtbarkeitsmodifizierer einer Klasse
- Sichtbarkeitsmodifizierer innerhalb einer Klasse

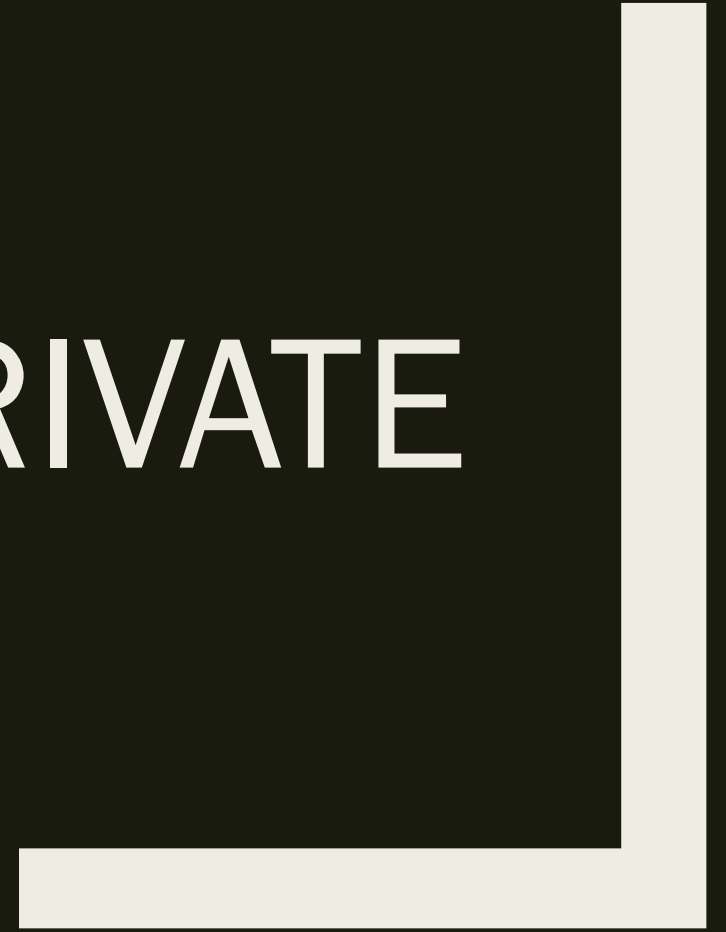
PUBLIC



Public

- deklariert öffentliche Typen und Eigenschaften
- Typen sind überall sichtbar
- jede Klasse und Unterklasse aus beliebigem Paket kann auf öffentliche Eigenschaften zugreifen
- Mit public deklarierten Methoden und Variablen sind sichtbar, wo die Klasse sichtbar ist.
- Eigenschaften einer unsichtbaren Klasse sind unsichtbar

PRIVATE



Private

- Die Klasse, die den Dateinamen bestimmt, kann nicht privat sein
- Methoden und Variablen sind nur innerhalb der eigenen Klasse sichtbar
- Innere Klassen, können auch auf private Eigenschaften der äußeren Klasse zugreifen
- Wird eine Klasse erweitert, sind die privaten Elemente für Unterklassen nicht sichtbar

PAKETSICHTBAR



Paketsichtbar

- ist die Standard-Sichtbarkeit und kommt ohne Modifizierer aus
- Paketsichtbare Typen und Eigenschaften sind nur für die Klassen aus dem gleichen Paket sichtbar, also weder für Klassen noch für Unterklassen aus anderen Paketen

PROTECTED



Protected

- **protected** hat eine Doppelfunktion:
 - *Es hat er die gleiche Bedeutung wie Paketsichtbarkeit*
 - *Es gibt die Elemente für Unterklassen frei*
 - *Dabei ist es egal, ob die Unterklassen aus dem eigenen Paket stammen*

SICHTBARKEITSMODIFIZIERER
EINE KLASSE



Sichtbarkeitsmodifizierer einer Klasse

- Eine Klasse kann lediglich zwei Sichtbarkeitsmodifizierer haben:
 - *Public*
 - *Default (Paketsichtbarkeit)*

Sichtbarkeitsmodifizierer einer Klasse

- Attribute und Methoden einer Klasse können vier Sichtbarkeitsmodifizierer haben:
 - *public*
 - *private*
 - *default* (*Paketsichtbarkeit*)
 - *protected*

Bedeutung der Sichtbarkeit

- **Klasse B sichtbar für die Klasse A:** Klasse A hat Zugriff auf Attribute und Methoden der Klasse B

- Es gibt zwei **Zugriffsmöglichkeiten:**
 1. *Methode einer Klasse hat Zugriff auf Attribute und Methoden (Member) einer anderen Klasse*
 2. *Subklasse erbt Attribute und Methoden der Oberklasse*

```
class Zoo {
    public String coolMethod() {
        return "Wow  baby";
    }
}
class Moo {
    public void useAZoo() {
        Zoo z = new Zoo();
        // If the preceding line compiles Moo has access
        // to the Zoo class
        // But... does it have access to the coolMethod()?
        System.out.println("A Zoo says, " + z.coolMethod());
        // The preceding line works because Moo can access the
        // public method
    }
}
```

Zugriff einer Klasse auf eine andere Klasse

- Methode einer Klasse hat über den Punkt Operator Zugriff auf Attribute und Methoden der anderen Klasse

```
class Zoo {
    public String coolMethod() {
        return "Wow  baby";
    }
}
class Moo extends Zoo {
    public void useMyCoolMethod() {
        // Does an instance of Moo inherit the coolMethod()?
        System.out.println("Moo says, " + this.coolMethod());
        // The preceding line works because Moo can inherit the
        // public method
        // Can an instance of Moo invoke coolMethod() on an
        // instance of Zoo?
        Zoo z = new Zoo();
        System.out.println("Zoo says, " + z.coolMethod());
        // coolMethod() is public, so Moo can invoke it on a Zoo
        //reference
    }
}
```

Zugriff über Vererbung

- Zugriff bedeutet: Subklasse erbt Attribute und Methoden der Oberklasse (Es ist als ob die Subklasse diese selbst implementieren würde)

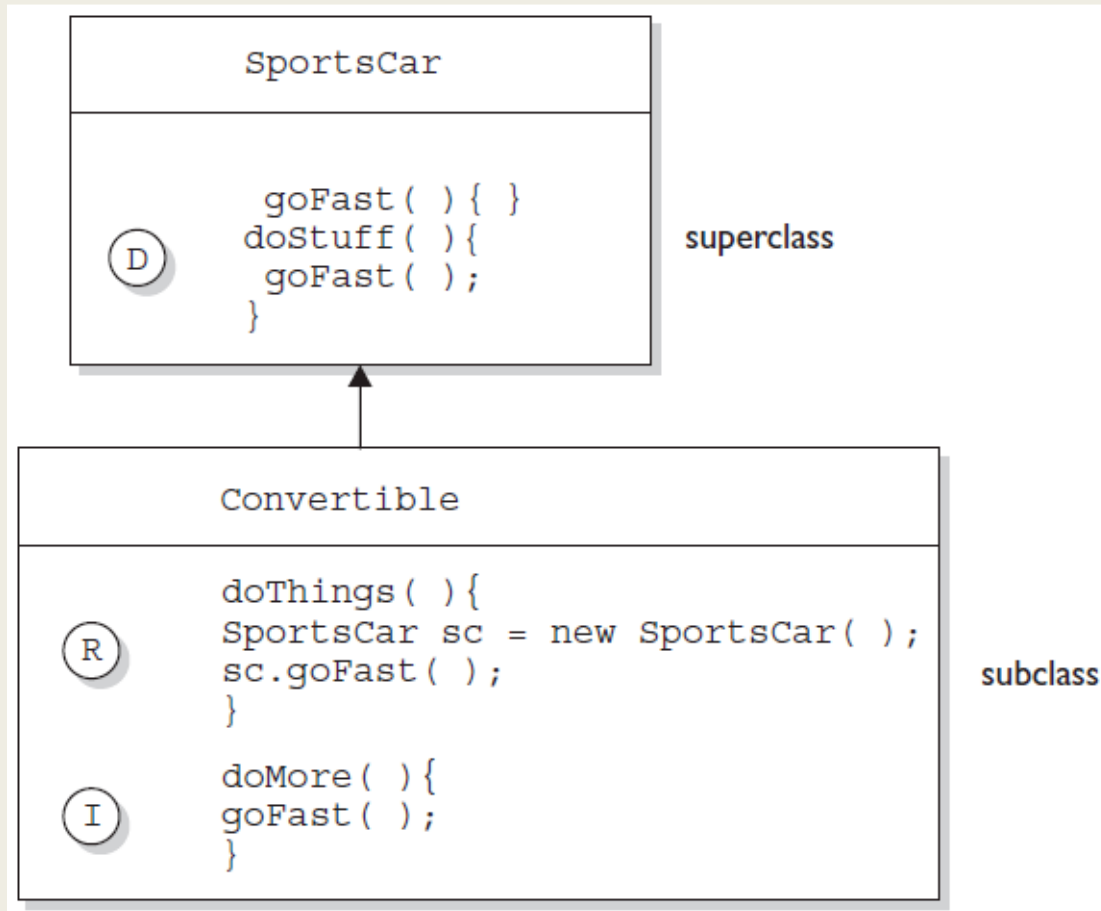
VERERBUNG VS. ZUGRIFF ÜBER REFERENZ



Vererbung vs. Zugriff über Referenz

- Liegen die Klassen im gleichen oder in unterschiedlichen Paketen?
- Ist Klasse B sichtbar für Klasse A?
(unsichtbare Klasse bedeutet unsichtbare Member)
- Unterschiedliche Kombinationen der Klassensichtbarkeit und Membersichtbarkeit

Vererbung vs. Zugriff über Referenz



Three ways to access a method:

- (D) Invoking a method declared in the same class
- (R) Invoking a method using a reference of the class
- (I) Invoking an inherited method

Vererbung vs. Zugriff über Referenz

Driver	
(R)	<pre>doDriverStuff(){ SportsCar car = new SportsCar(); car.goFast(); }</pre>
(R)	<pre>Convertible con = new Convertible(); con.goFast(); }</pre>

Three ways to access a method:

- (D) Invoking a method declared in the same class
- (R) Invoking a method using a reference of the class
- (I) Invoking an inherited method

ATTRIBUTE UND METHODEN



PUBLIC

again



Public - Attribute und Methoden (Members)

- public – Attribute und Methoden (Member) sind aus jeder Klasse und Paket aus sichtbar falls die Klasse selber auch sichtbar ist
- Goo kann auf die Methode testIt() der Klasse Sludge zugreifen, da sowohl die Klasse Sludge und Methode testIt() öffentlich sind

Public - Attribute und Methoden (Members)

```
package book;
import cert.*; // Import all classes in the cert package
class Goo {
    public static void main(String[] args) {
        Sludge o = new Sludge();
        o.testIt();
    }
}
```

```
package cert;
public class Sludge {
    public void testIt() { System.out.println("sludge"); }
}
```


public Members und Vererbung

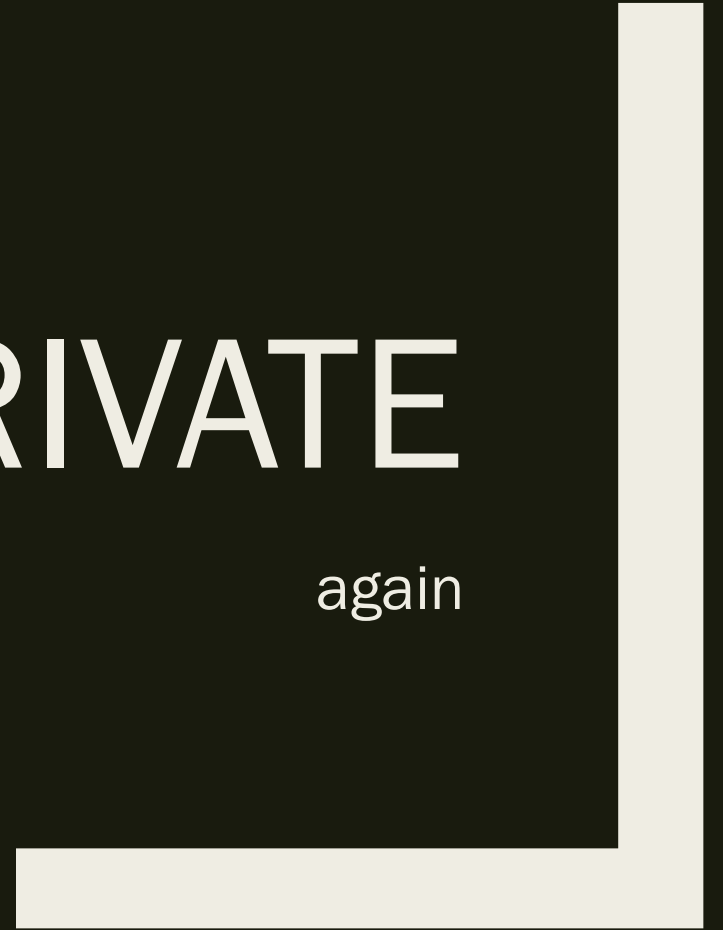
- Vererbung ist paketunabhängig
- Zugriff nicht über Punktoperator, Methode gehört zur Klasse

```
package cert;  
public class Roo {  
    public String doRooThings() {  
        // imagine the fun code that goes here  
        return "fun";  
    }  
}
```

```
package notcert;    //Not the package Roo is in  
import cert.Roo;  
class Cloo extends Roo {  
    public void testCloo() {  
        System.out.println(doRooThings());  
    }  
}
```

PRIVATE

again



private - Attribute und Methoden (Members)

- **private** – Attribute und Methoden (Member) sind nur in der Klasse sichtbar, in der sie definiert worden sind
- Es ist als ob die Methode doRooThings() nicht existieren würde

private - Attribute und Methoden (Members)

```
package cert;
public class Roo {
    private String doRooThings() {
        // imagine the fun code that goes here, but only the Roo
        // class knows
        return "fun";
    }
}
```

```
package notcert;
import cert.Roo;
class UseARoo {
    public void testIt() {
        Roo r = new Roo(); //So far so good; class Roo is public
        System.out.println(r.doRooThings()); //Compiler error!
    }
}
```

```
cannot find symbol
symbol   : method doRooThings()
```

private Members und Vererbung

- Privates Attribut oder Methode kann nicht vererbt werden
- Die Deklaration einer passenden Methode doRooThings() in der Subklasse ist möglich – diese überschreibt **nicht** die Methode der Oberklasse

```
package cert;  
public class Roo {  
    private String doRooThings() {  
        // imagine the fun code that goes here, but no other class  
        // will know  
        return "fun";  
    }  
}
```

private Members und Vererbung

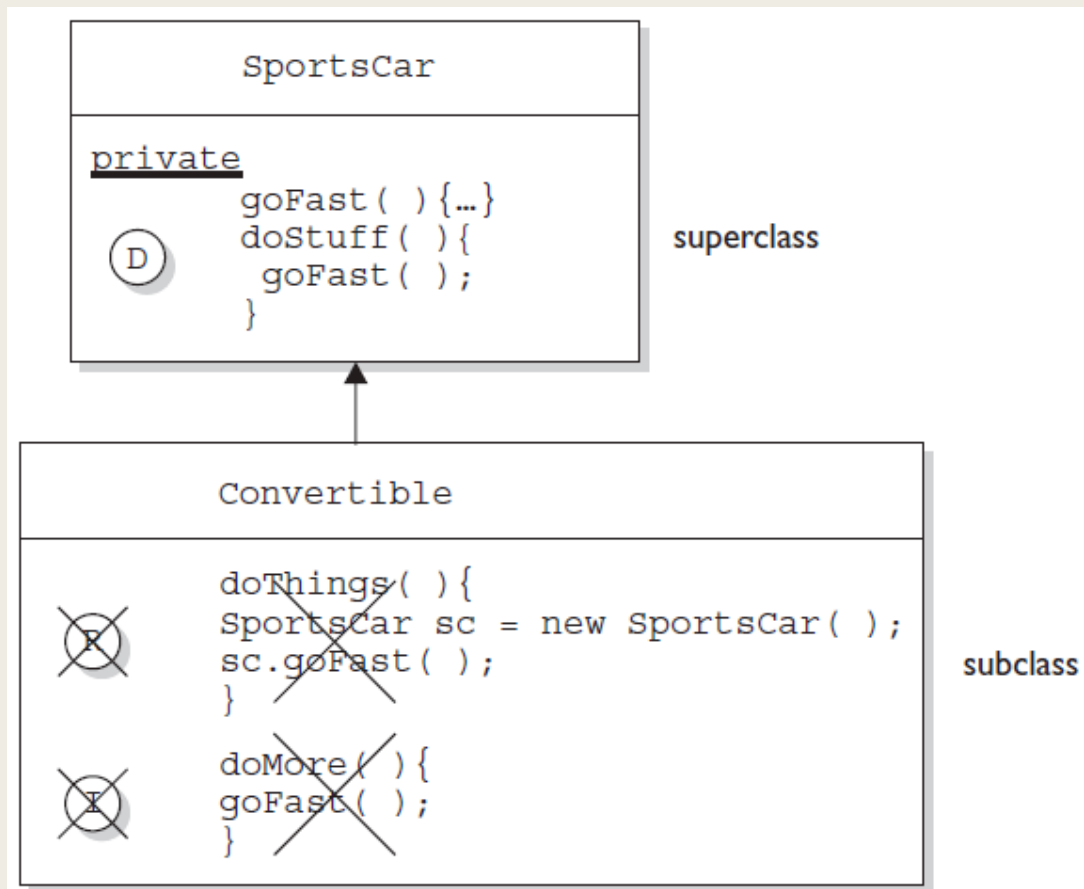
- Kindsklasse Cloo von Roo hat auch keinen Zugriff auf die Methode doRooThings() der Klasse Roo

```
package cert;                //Cloo and Roo are in the same package
class Cloo extends Roo {    //Still OK, superclass Roo is public
    public void testCloo() {
        System.out.println(doRooThings()); //Compiler error!
    }
}
```

```
%javac Cloo.java
Cloo.java:4: Undefined method: doRooThings()
    System.out.println(doRooThings());
1 error
```

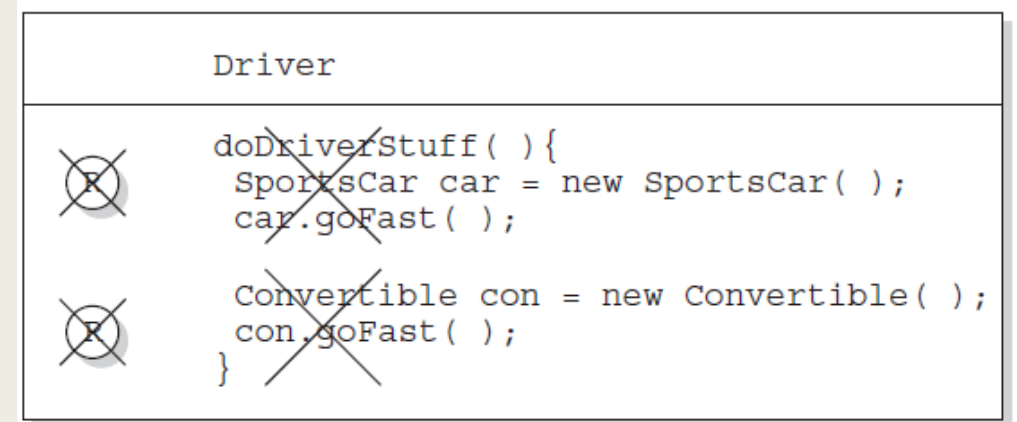
private Attribute

- Attribute werden als private deklariert Hintergrund: Kapselung bzw. Data Protection (Encapsulation)



Three ways to access a method:

- (D) Invoking a method declared in the same class
- (R) Invoking a method using a reference of the class
- (I) Invoking an inherited method



PROTECTED UND PAKETSICHERBARKEIT

again

protected und Paketsichtbarkeit (default)

- **protected** und **Paketsichtbarkeit** sind sich sehr ähnlich mit einem entscheidendem Unterschied:
 1. *auf ein default Member kann nur dann zugegriffen werden, wenn die zugreifende Klasse sich im gleichen Paket befindet*
 2. *auf ein protected Member kann durch Vererbung zugegriffen werden, der Ort (Paket) der Subklasse spielt dabei keine Rolle*

protected und Paketsichtbarkeit (default)

- Methode testIt() der Klasse OtherClass paketsichtbar
- AccessClass befindet sich nicht im Paket certification

```
package certification;
public class OtherClass {
    void testIt() {    // No modifier means method has default
                      // access
        System.out.println("OtherClass");
    }
}
```

```
package somethingElse;
import certification.OtherClass;
class AccessClass {
    static public void main(String[] args) {
        OtherClass o = new OtherClass();
        o.testIt();
    }
}
```

```
No method matching testIt() found in class
certification.OtherClass.    o.testIt();
```

protected und Paketsichtbarkeit (default)

- default access (Paketsichtbarkeit): package restriction
- protected: package + kids
- Klasse mit protected Attribute und Methoden:
 - *Für Attribute und Methoden besteht Paketsichtbarkeit mit Ausnahme der Subklassen, die sich außerhalb des Pakets befinden*
 - *Zugriff der Subklassen auf protected Attribute und Methoden ist über den Punktoperator nicht möglich*

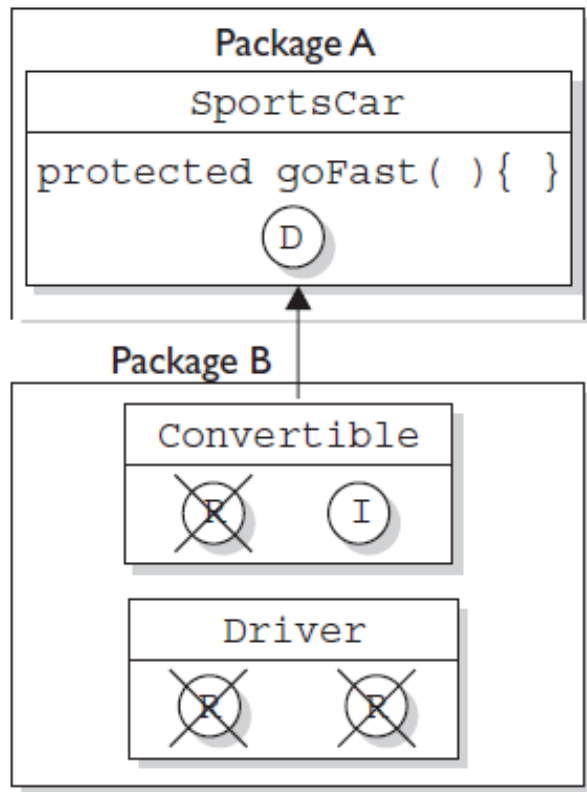
protected Attribute und Methoden

■ Zugriff über Referenz - Fehlermeldung

```
%javac -d . other/Child.java
other/Child.java:9: x has protected access in certification.Parent
System.out.println("X in parent is " + p.x);
                                     ^
1 error
```

protected Attribute und Methoden

If goFast() is protected



(D)

```

goFast( ) {
doStuff( ) {
    goFast( );
}
    
```

Where goFast
is *Declared* in the
same class.

(I)

```

doMore( ) {
    goFast( );
}
    
```

Invoking the
goFast()
method
Inherited from
a superclass.

(R)

```

doThings( ) {
    SportsCar sc = new SportsCar( );
    sc.goFast( );
}
    
```

Invoking goFast() using a *Reference* to the
class in which goFast() was declared.

Three ways to access a method:

- (D) Invoking a method declared in the same class
- (R) Invoking a method using a reference of the class
- (I) Invoking an inherited method

default Attribute und Methoden

■ Zugriff über Referenz - unterschiedliche Pakete

```
package certification;  
public class Parent {  
    int x = 9; // No access modifier, means default  
              // (package) access  
}
```

```
package other;  
import certification.Parent;  
class Child extends Parent {  
    public void testIt() {  
        System.out.println("x is " + x);  
  
        Parent p = new Parent();  
  
        System.out.println("X in parent is " + p.x);  
    }  
}
```

```
Child.java:4: Undefined variable: x  
    System.out.println("Variable x is " + x);  
1 error
```


default Attribute und Methoden

■ Zugriff über Referenz - gleiches Paket

```
package certification;
public class Parent{
    int x = 9; // default access
}
```

```
package certification;
class Child extends Parent{
    static public void main(String[] args) {
        Child sc = new Child();
        sc.testIt();
    }
    public void testIt() {
        System.out.println("Variable x is " + x); // No problem;
    }
}
```

LOKALE VARIABLEN UND SICHTBARKEITSMODIFIZIERER



Lokale Variablen und Sichtbarkeitsmodifizierer

- Wird niemals kompilieren

```
class Foo {  
    void doStuff() {  
        private int x = 7;  
        this.doMore(x);  
    }  
}
```

- Lokale Variablen können nur mit final modifiziert werden.

Zusammenfassung Sichtbarkeitsmodifizierer

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, <i>through inheritance</i>	No	No
From any non-subclass class outside the package	Yes	No	No	No