

Inheritance 1

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at

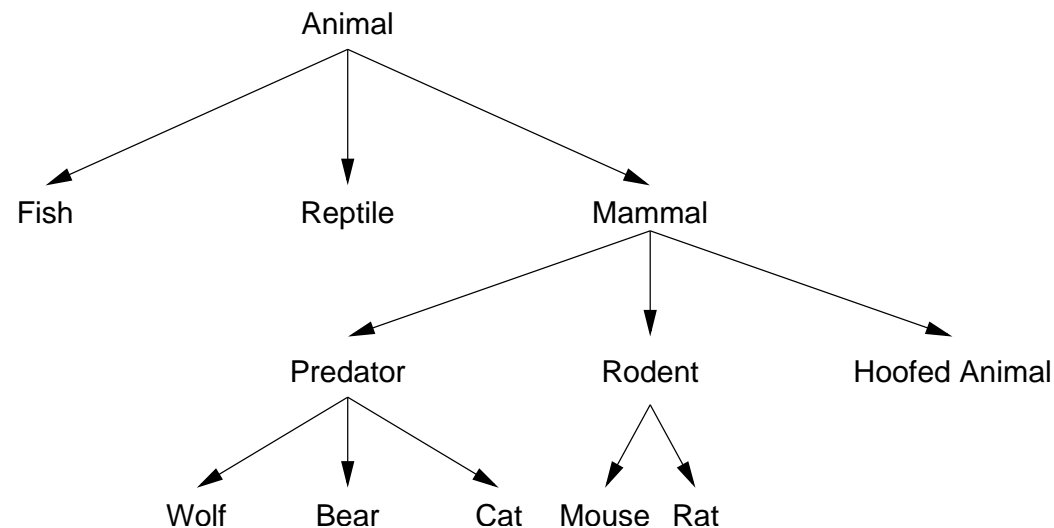
<http://www.risc.jku.at>

Fundamental Ideas

Classes

Classes represent collections of uniform objects.

- In reality, objects come in **variants**.
 - Often the variants can be hierarchically classified.



Fishes and mammals are both animals.

Common and Distinguishing Properties

Whether two objects belong to a class depends on how close we look.

- “A wolf and a mouse are both mammals.”
 - Common characteristics: both breastfeed their offspring.
- “A wolf is a predator while a mouse is a rodent.”
 - A wolf eats other animals.
 - A mouse eats corn.
- “Predator” and “mouse” have a common ancestor “mammal”.
 - The classes share some properties but differ in other properties.

Java offers a similar organization of classes.

Shop Example

An internet shop offers books and CDs.

- Books:
 - Article number, title, price.
 - Author, publisher, ISBN number.
- CDs:
 - Article number, title, price.
 - Interpreter, list of songs.

A shopping cart shall list number, title, and price of articles.

Common Functionality

The common functionality may be extracted to a class.

```
public class Article
{
    public int number;
    public String title;
    public int price;

    public Article (...) { ... }

    public int getNumber() { ... }
    public String getTitle() { ... }
    public int getPrice() { ... }
}
```

Books and CDs are special cases of articles.

Inheritance

Special functionality may be added to the common functionality.

```
public class Book extends Article
{
    public String author;
    public String publisher;
    public String ISBN;

    public Book(...) { ... }

    public String getAuthor() { ... }
    public String getPublisher() { ... }
    public String getISBN() { ... }
}
```

Class Book inherits fields and methods of class Article.

Inheritance

Child classes inherit from parent classes.

```
public class Subclass extends Superclass implements Interface, ...  
{  
    ...  
}
```

- *Subclass* is the **child** of *Superclass*.
 - *Superclass* is the **parent** of *Subclass*.
 - **Single inheritance**: a child class may have only one parent.
 - * A class may implement **multiple** interfaces.
- *Subclass* **inherits** all fields and methods of *Superclass*.
 - Fields and methods can be used as if they were defined locally within *Subclass*.
 - *Subclass* may add additional fields and methods.

Inheritance

A class may have multiple children.

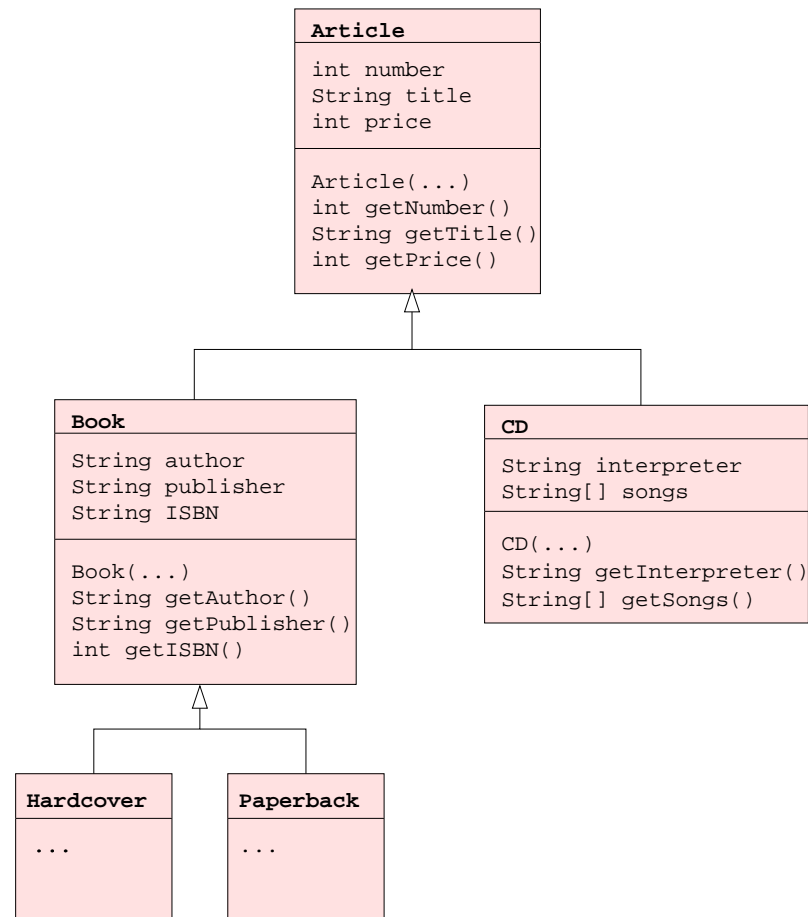
```
public class CD extends Article
{
    public String interpreter;
    public String[] songs;

    public CD(...) { ... }

    public String getInterpreter() { ... }
    public String[] getSongs() { ... }
}
```

Class CD inherits fields and methods of class Article.

Inheritance Hierarchy



Interface Inheritance

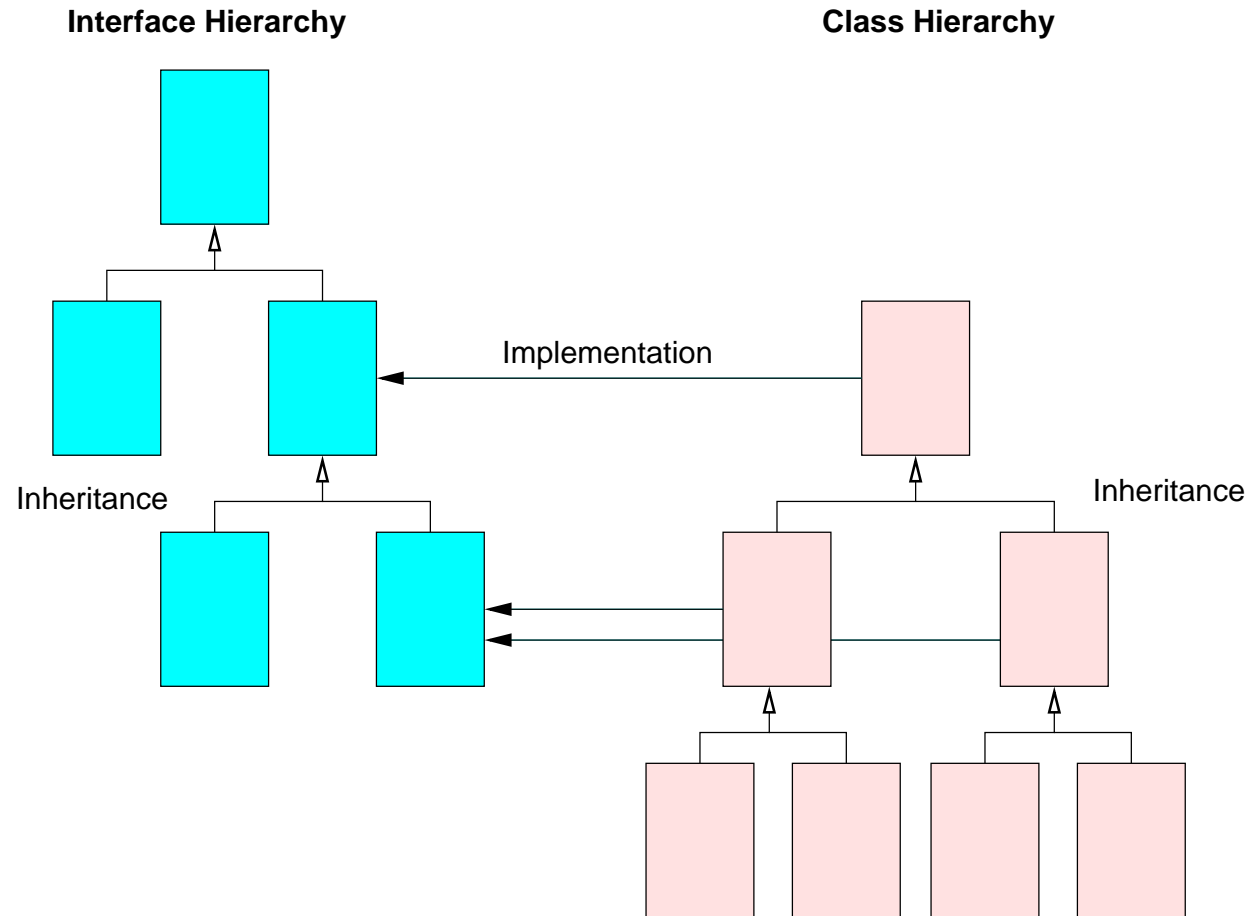
Also an interface may inherit from another interface.

```
public interface I extends J
{
    ...
}
```

- Interface *I* inherits methods and constants of interface *J*.
 - An interface **cannot** inherit from a class.
 - A class **cannot** inherit from an interface.

Classes and interfaces form separate inheritance hierarchies.

Interface and Class Inheritance



Constructors in Parent Classes

```
public class Article
{
    public int number;
    public String title;
    public int price;

    public Article(int n, String t, int p)
    {
        number = n;
        title = t;
        price = p;
    }
}
```

A parent class may have constructors.

Constructors in Child Classes

```
public class Book extends Article
{
    public String author;
    public String publisher;
    public String ISBN;

    public Book(int n, String t, int p, String a, String pb, String i)
    {
        number = n; title = t; price = p;
        author = a;
        publisher = pb;
        ISBN = i;
    }
}
```

A child class has its own constructors.

Calls of Constructors of Parent Classes

```
public class Book extends Article
{
    public String author;
    public String publisher;
    public String ISBN;

    public Book(int n, String t, int p, String a, String pb, String i)
    {
        super(n, t, p);
        author = a;
        publisher = pb;
        ISBN = i;
    }
}
```

A constructor may call with `super` a parent constructor.

Code Sharing

Inheritance reduces the amount of code to be written.

- Multiple subclasses inherit from the same superclass.
 - Common functionality is defined only **once** in the superclass.

- **General rule:**

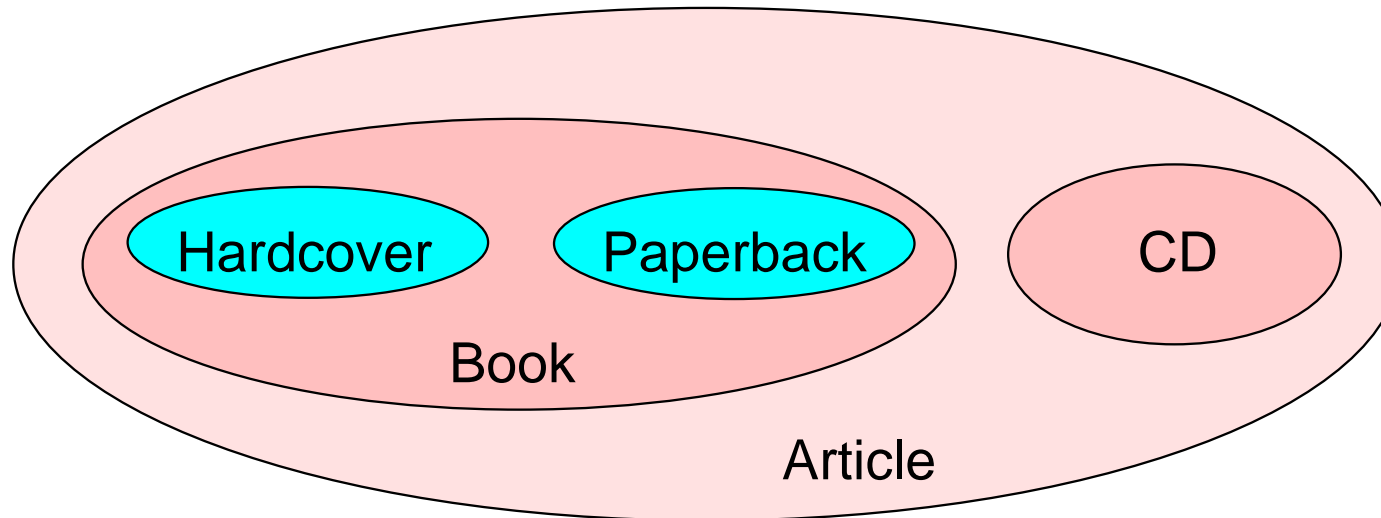
Whenever there are two or more classes that share common functionality, this functionality should be put in a separate class; from this class, the other classes can be derived by inheritance.

Avoid code duplication by inheritance.

Dynamic Types and Genericity

Is-Relationship

Every object of a subclass **is** also an object of the superclass.



An object of type **Book** is also an object of type **Article**.

Type Compatibility

A subclass is compatible with the super class.

- **General rule:**

Every program that is able to work with objects of the superclass can automatically also work with objects of the subclass.

- **Example: internet shop.**

- Write program that works with objects of type `Article`.
 - * Place articles in shopping cart, print contents of cart, compute total price of order.
- Later implement concrete articles as subclasses of `Article`.
 - * `Book`, `CD`, `Video`, ...
- Program can **immediately** work with these articles.
 - * Program need **not** be modified.

First write the general functionality, then the more special one.

Object Assignments

A variable whose type is a class may refer to an object of a subclass.

- A Book is an Article and a CD is an Article.

```
Article a = new Book(...);  
a = new CD(...);
```

- A Article variable may hold a Book or a CD.
 - Can only refer to the Article fields and methods.

```
String t = a.getTitle();  
int p = a.getPrice();
```

- May check the actual type with instanceof:

```
if (a instanceof Book)  
{  
    Book b = (Book)a;  
    ...  
}
```

Object Assignments

```
public class D extends C
{
    ...
}
```

```
D d = ...
C c = d;
```

```
C c = ...;
if (c instanceof D)
{
    D d = (D)c;
    ...
}
```

Static Types and Dynamic Types

An object variable has two kinds of types.

1. The **static** type:

- The type with which the variable has been declared.
- Determines to which fields and methods we can refer.
- `Article a = new Book(...);`

2. The **dynamic** type:

- The type of the object to which the variable refers.
- Determines which methods are actually called (see later).
- `Article a = new Book(...);`

The dynamic type is always a subtype of the static type.

Generic Methods

An object variable or parameter can refer to objects of different types.

```
public static void printInfo(Article a)
{
    int number = a.getNumber();
    String title = a.getTitle();
    int price = a.getPrice();
    System.out.print("Article '" + title);
    System.out.print( "' (" + number + ") costs ");
    System.out.println(price/100 + "." + price%100 + " Euro.");
}
```

A generic method may operate on objects of different types.

Generic Methods

A generic method may be invoked with objects of different types.

```
public class Book extends Article { ... }  
public class CD extends Article { ... }
```

```
Book book = new Book(...);  
CD cd = new CD(...);  
printInfo(book);  
printInfo(cd);
```

The argument may be any subtype of the parameter type.

The Object Class

Every class is a (direct or indirect) subclass of `Object`.

```
public class Class
{
    ...
}

public class Class extends Object
{
    ...
}
```

`Object` is a class of package `java.lang`.

A Generic Collection Class

Object may serve as a placeholder for any class.

```
public class Stack
{
    public int N = 100;
    public Object[] stack = new Object[N];
    public int n = 0;

    public void push(Object o)
    { if (n == N) resize();
      stack[n] = o;
      n++;
    }

    public Object pop() { n--; return stack[n]; }
}
```

Application

Any object type may be stored in the generic collection class.

```
Stack s = new Stack();  
s.push(new Integer(2));  
s.push(new Integer(3));  
Integer i = (Integer)(s.pop());
```

The result of a selector function is `Object` as well.

Application

Different object types may be stored in a generic collection.

```
Stack s = new Stack();
s.push(new Integer(2)); s.push("hello");
Object o = s.pop();
if (o instanceof Integer)
{
    Integer i = (Integer)o;
    ...
}
else if (o instanceof String)
{
    String s = (String)o;
    ...
}
```

See the Java standard class `Stack`.