Java Programming
Module
OO concepts and Inheritance

DHBW Duale Hochschule Baden-Württemberg Mannheim

Prof. Dr. Holger D. Hofmann

Object-oriented Concepts: Abstraction

- Separation of concept and realisation
- Abstraction of common traits to from common behavior (code)
- Abstraction also includes joint sets of properties/traits:
  - Example: A person and a student have many things in common
- Behavior and properties can be abstracted from different persons to form a "template person"
- Abstraction in Object-orientation means to build a model

Object-oriented Concepts: Encapsulation

- Separation of abstraction from implementation
  - only "visible" attributes/methods can be used
- In software development: interface and implementation
  - Java interfaces will be covered later
- Realisation of "black-box" concept
  - -> black-box reuse
- Realisation or implementation details are unknown to users
- Data is encapsulated within Objects

Encapsulation in Java: Modifiers

The visibility of classes, attributes and methods can be defined using the statements

- public – visible to everybody
- protected – visible to a class itself, derived classes, and classes from within the same package
- private – visible only with a class itself
- [none] – "package scoped", i.e., are visible within the same package

Accessing Private Attributes/Methods

- Objects may access private members of other objects belonging to the same class.

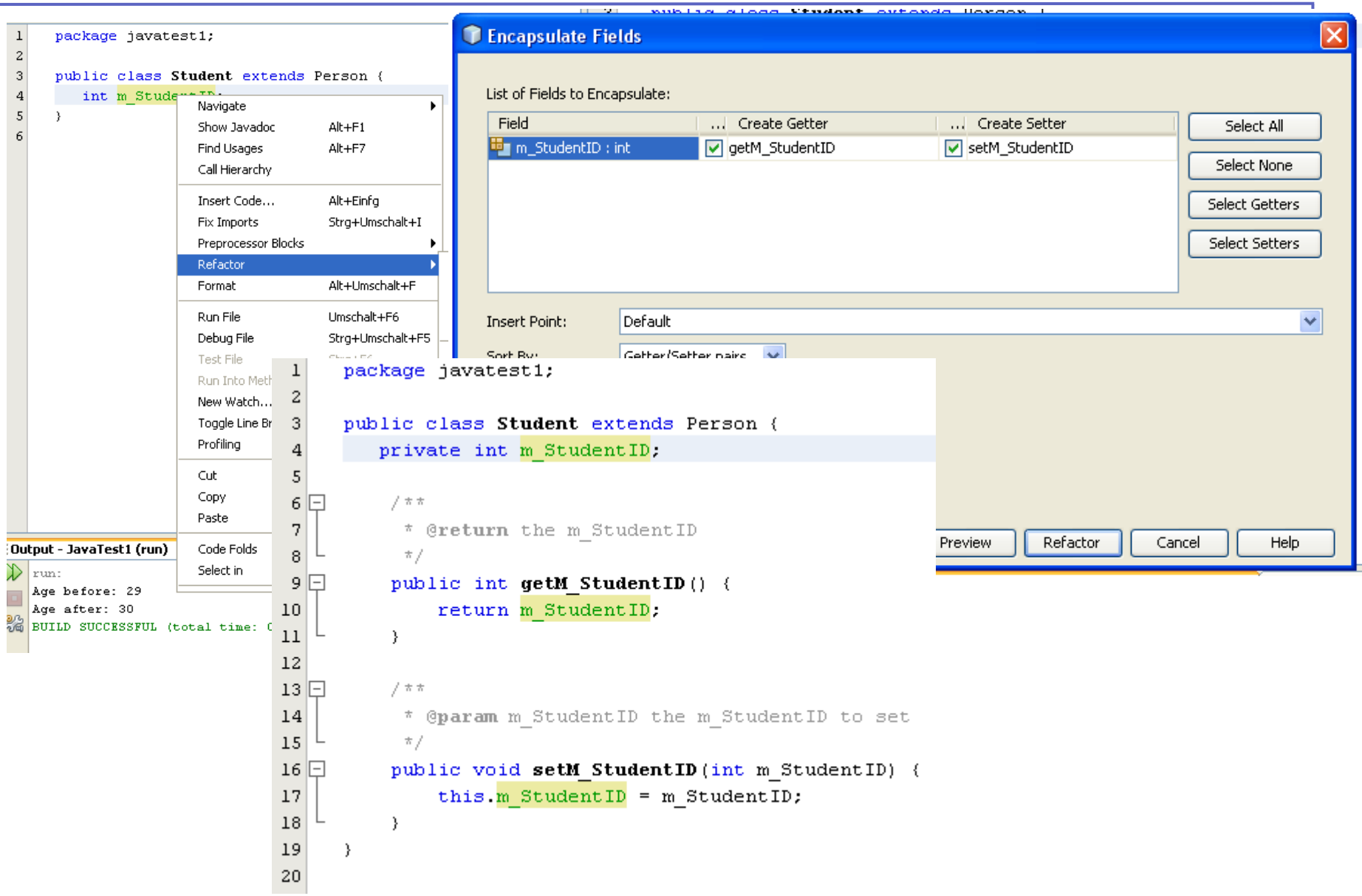
```
001 /* Listing0806.java */
002
003 public class Listing0806
004 {
005   public static void main(String[] args)
006   {
007     ClassWithPrivateA a1 = new ClassWithPrivateA(7);
008     ClassWithPrivateA a2 = new ClassWithPrivateA(11);
009     a2.setOtherA(a1, 999);
010     System.out.println("a1 = " + a1.toString());
011     System.out.println("a2 = " + a2.toString());
012   }
013 }
014
015 class ClassWithPrivateA
016 {
017   private int a;
018
019   public ClassWithPrivateA(int a)
020   {
021     this.a = a;
022   }
```

Accessing Private Attributes/Methods

```
023
024   public void setOtherA(ClassWithPrivateA other, int newvalue)
025   {
026     other.a = newvalue;
027   }
028
029   public String toString()
030   {
031     return "" + a;
032   }
033 }
```

- Output:
  - a1 = 999
  - a2 = 11

Save Coding Work: Getter/Setter Generation



Object-oriented Concepts: Relationship

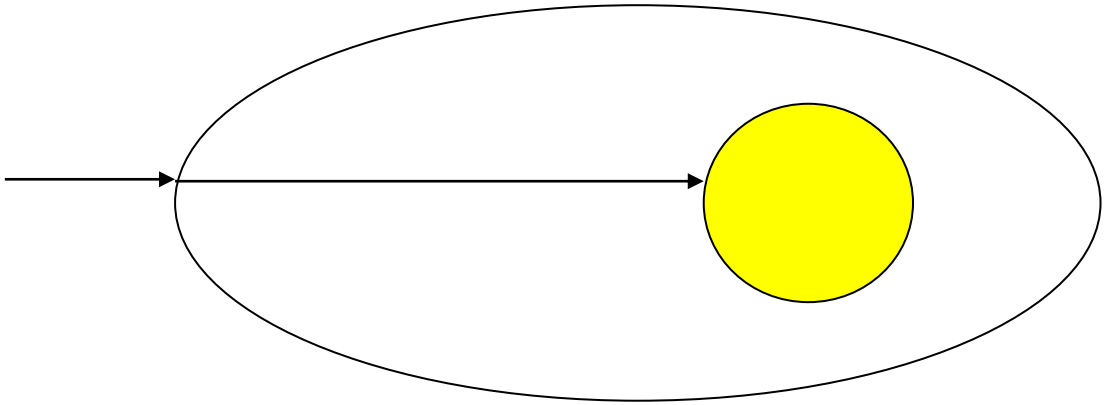
- Classes can have relationships to other classes
  - For example, class A can have an attribute of class (=type) B
- Objects can have relationships to other objects
  - For example, a Person object can hold many references to other Person objects via an Array "Friends"
- Types of relationships:
  - Is-a relationship (Subclass/Superclass)
  - Part-of relationship (Containment)
  - Usage or caller relationship ("Delegation")

Object-oriented Concepts: Aggregation and Composition

- Is realised via the part-of relationship
- Aggregation can be the modelling of physical containment (e.g., a car contains an engine, 5 tyres, ...) -> Composition

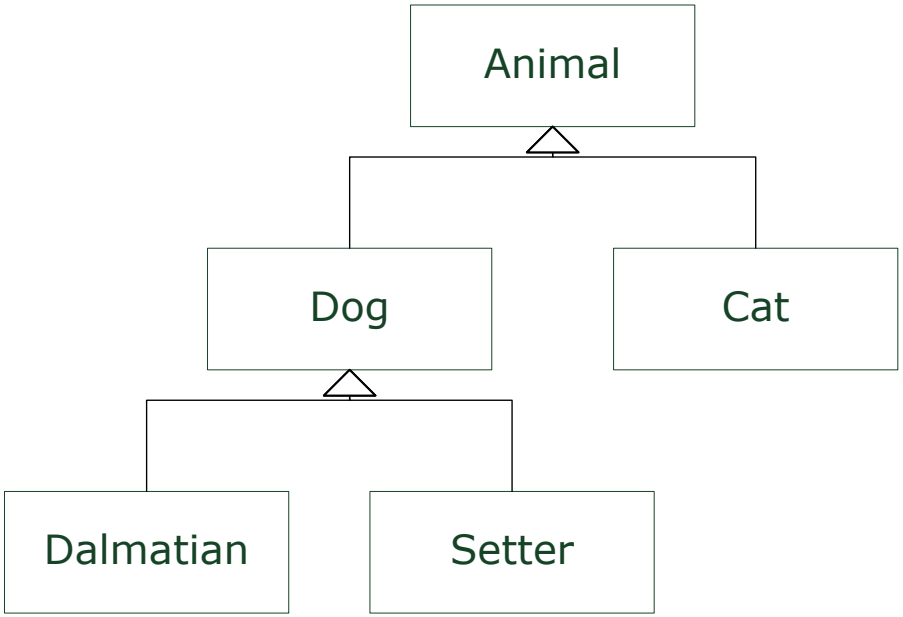
  Or

- Simple usage/delegation scenario -> Aggregation
  - Containing object makes calls to contained objects

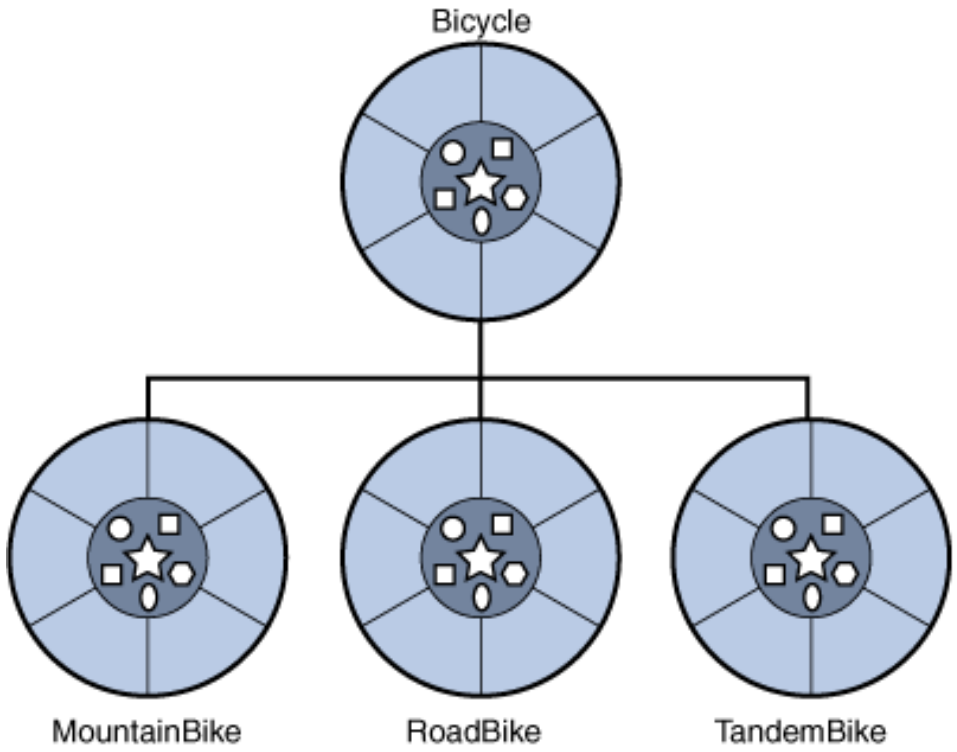


Object-oriented Concepts: Generalisation and Specialisation

- Generalisation and specialisation are orthogonal concepts and are realised via the is-a relation
  - One class B can be derived from another class A (inheritance)
  - B may be a specialisation and thus may contain more detailed data/methods
  - In Java, a class can only be derived from one "base class" (single implementation inheritance)



Inheritance




```
1.class MountainBike extends Bicycle {

2.// new fields and methods defining a mountain bike would go
  // here

3.}
```

Source: Sun Java Tutorial

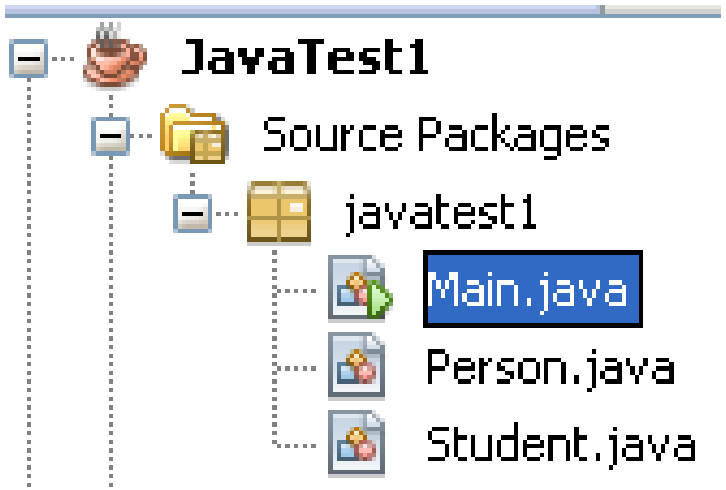
Inheritance Example

```
001 package javatest1;
002
003 public class Student extends Person {
004    int m_StudentID;
005    int getStudentID(){
006     return m_StudentID;}
007 }
```

Run Program:

java javatest1.main

Netbeans Project View Example



Inheritance in Java

```
public class Student extends Person
```

- Class inheritance results in reusing
  - the Signature and
  - the Implementation

  of the base class

- Inheritance builds an is-a relationship (a student is-a person)

Exercises 3.*