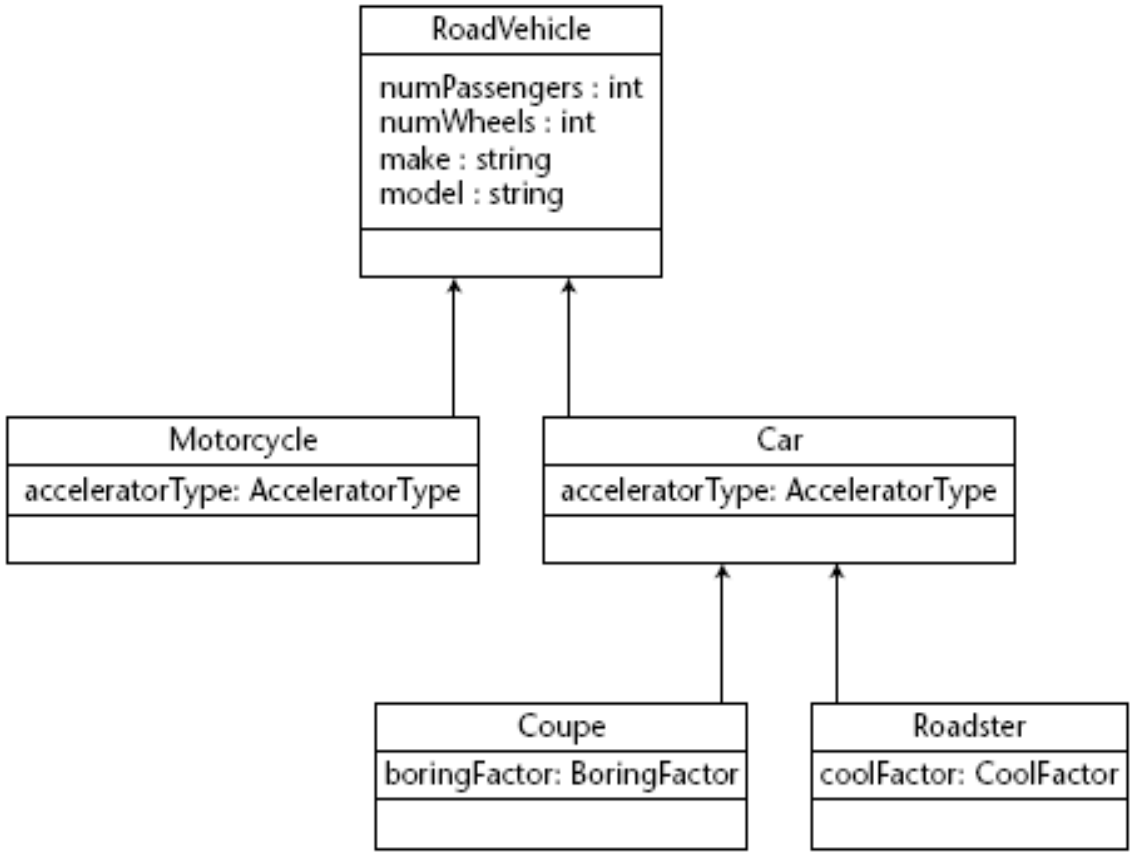
Advanced Persistency

Inheritance

Mapping inheritance



SINGLE TABLE PER CLASS

Id	numPass	numWheels	make	model
1	6	2	HORSECART	NULL

Id	numPass	numWheels	make	model	acceleratorType
2	1	2	HONDA	HRC7	THROTTLE

etc.

Problems with polymorphism – how do you find "all RoadVehicles that have less than 3 passenger?"

SINGLE TABLE PER CLASS HIERARCHY

Id	numPass	numWheels	make	model	DISC	acceleratortype	BoringFactor	CoolFactor
1	6	2	HORSECART	NULL	ROADVEHICLE	NULL	NULL	NULL
2	1	2	HONDA	HRC7	MOTORCYCLE	THROTTLE	NULL	NULL
3	4	4	FIAT	PUNTO	CAR	PEDAL	NULL	NULL
4	2	4	FERRARI	F70	COUPE	PEDAL	1	NULL
5	2	4	FORD	KA	ROADSTER	PEDAL	NULL	1

- **Space inefficiency**
- **Impossible to set "NON-NULL" constraints on fields of the subclasses.**

JOINED TABLES

RoadVehicle

Id	DTYPE	numPass	numWheels	make	model
1	ROADVEHICLE	6	2	HORSECART	NULL
2	MOTORCYCLE	1	2	HONDA	HRC7
3	CAR	4	4	FIAT	PUNTO
4	COUPE	2	4	FERRARI	F70
5	ROADSTER	2	4	FORD	KA

Car

Id	acceleratortype
3	PEDAL
4	PEDAL
5	PEDAL

Coupe

Id	boringFactor
4	1

Many joins in a deep inheritance hierarchy – time inefficiency.

The base class

```
package examples.entity.single_table;
// imports go here
@Entity(name="RoadVehicleSingle")
@Table(name="ROADVEHICLE") //optional, it's the default
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISC",
   discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("ROADVEHICLE")
// @Inheritance(strategy=InheritanceType.JOINED)
public class RoadVehicle implements Serializable {
    public enum AcceleratorType {PEDAL,THROTTLE};
    @Id
    protected int id;
    protected int numPassengers;
    protected int numWheels;
    protected String make;
    protected String model;
    public RoadVehicle() {
        id = (int) System.nanoTime();
    }
    // setters and getters go here
    ...
}
```

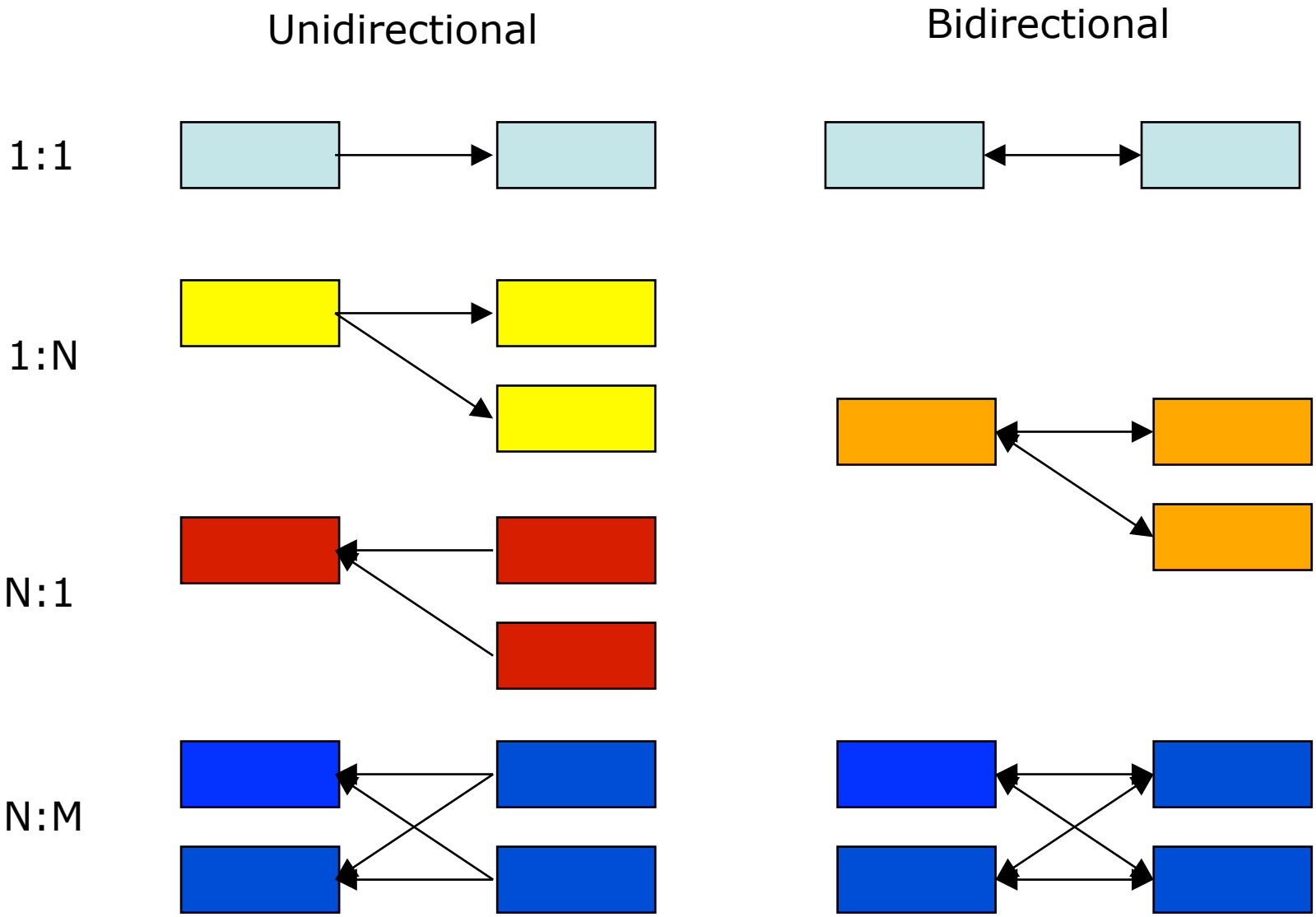
The derived class

```
package examples.entity.single_table;
// imports go here
@Entity
@DiscriminatorValue("MOTORCYCLE") //not needed for joined
public class Motorcycle extends RoadVehicle implements
   Serializable {
    public final AcceleratorType acceleratorType
      =AcceleratorType.THROTTLE;
    public Motorcycle() {
        super();
        numWheels = 2;
        numPassengers = 2;
    }
}
```

Advanced Persistency

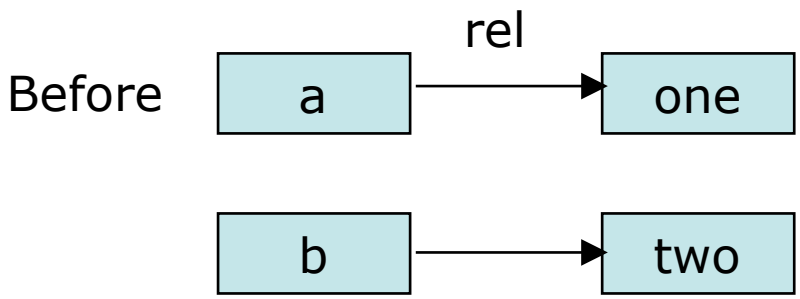
Relationships

Multiplicity and Directionality – 7 types

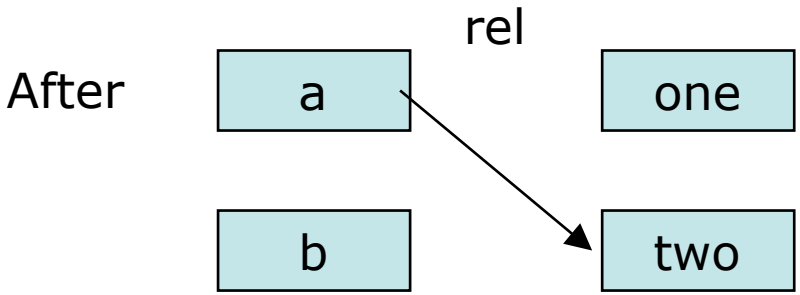


Watch out for side effects!

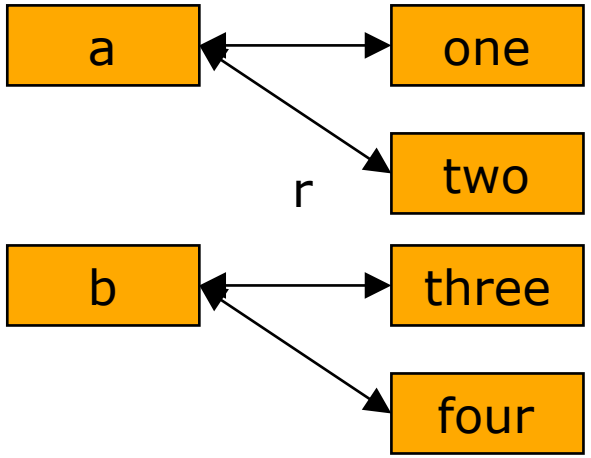
Let rel be a 1:1 relationship



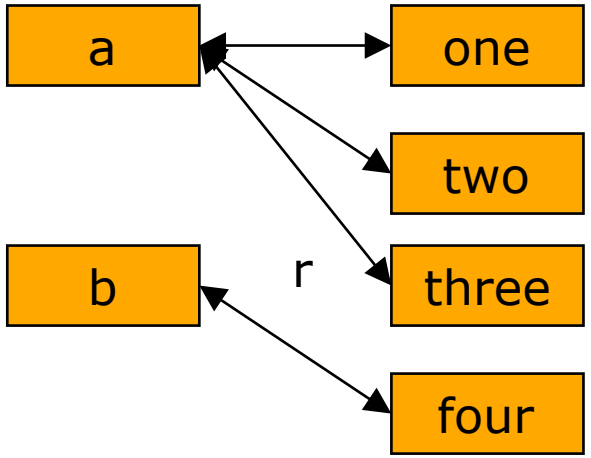
a.setRel(two)



Let r be a 1:N relationship

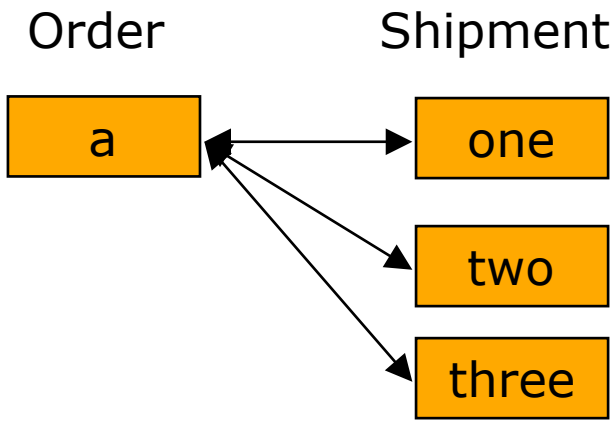


a.setR(three)



Cascade-delete

When we delete “a”,
should also one,two e three
be canceled?



Relation – 1:1 unidir – “from”

```
@Entity(name=”OrderUni”)
public class Order implements Serializable {
    private int id;
    private String orderName;
    private Shipment shipment;
    public Order() { id = (int)System.nanoTime(); }
    @Id
    public int getId() { return id; }
    public void setId(int id) {
    this.id = id;
    }
    ...
    // other setters and getters go here
    ...
    @OneToOne(cascade={CascadeType.PERSIST})
    public Shipment getShipment() {
        return shipment;
    }
    public void setShipment(Shipment shipment) {
        this.shipment = shipment;
    }
}
```

Relation – 1:1 unidir – “to”

```
...
@Entity(name=”ShipmentUni”)
public class Shipment implements Serializable {
    private int id;
    private String city;
    private String zipcode;
    public Shipment() { id = (int)System.nanoTime(); }
    @Id
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    ...
    // other setters and getters go here
}
```

Relation – 1:1 unidir – client

```
...
@Stateless
public class OrderShipmentUniBean implements OrderShipment {
    @PersistenceContext
    EntityManager em;
    public void doSomeStuff() {
        Shipment s = new Shipment();
        s.setCity("Austin");
        s.setZipcode("78727");
        Order o = new Order();
        o.setOrderName("Software Order");
        o.setShipment(s);
        em.persist(o);
    }
    public List getOrders() {
        Query q = em.createQuery("SELECT o FROM OrderUni o");
        return q.getResultList();
    }
}
```

Relation – 1:1 bidir – “to”

```
...
@Entity(name=”ShipmentUni”)
public class Shipment implements Serializable {
    private int id;
    private String city;
    private String zipcode;
    private Order order;
    public Shipment() { id = (int)System.nanoTime(); }
    @Id
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    ...
    // other setters and getters go here
    ...
    @OneToOne(mappedBy=”shipment”)
    // shipmentproperty from the Order entity
    public Order getOrder() {
        return order;
    }
    public void setOrder(Order order) {
        this.order = order;
    }
}
```

Relation – 1:1 bidir – client

```
...
@Stateless
public class OrderShipmentUniBean implements OrderShipment {
    @PersistenceContext
    EntityManager em;
    public void doSomeStuff() {
        Shipment s = new Shipment();
        s.setCity("Austin");
        s.setZipcode("78727");
        Order o = new Order();
        o.setOrderName("Software Order");
        o.setShipment(s);
        em.persist(o);
    }
    public List getOrders() {
        Query q = em.createQuery("SELECT o FROM OrderUni o");
        return q.getResultList();
    }
    ..
    public List getShipments() {
        Query q = em.createQuery("SELECT s FROM Shipment s");
        return q.getResultList();
    }
}
```

Relation – 1:N unidir – “from”

```
...
@Entity(name=”CompanyOMUni”)
public class Company implements Serializable {
    private int id;
    private String name;
    private Collection<Employee> employees;
    ...
    // other getters and setters go here
    // including the Id
    ...
    @OneToMany(cascade={CascadeType.ALL},fetch=FetchType.EAGER)
        public Collection<Employee> getEmployees() {
        return employees;
    }
    public void setEmployees(Collection<Employee> employees) {
        this.employees = employees;
    }
}
```

Relation – 1:N unidir – “to”

```
...
@Entity(name=”EmployeeOMUni”)
public class Employee implements Serializable {
    private int id;
    private String name;
    private char sex;
    ...
    // other getters and setters go here
    // including the Id
    ...
}
```

Relation – 1:N unidir – client

```
Company c = new Company();
c.setName("M*Power Internet Services, Inc.");Collection<Employee>
   employees = new ArrayList<Employee>();
Employee e = new Employee();
e.setName("Micah Silverman"); e.setSex('M'); employees.add(e);
e = new Employee();
e.setName("Tes Silverman"); e.setSex('F'); employees.add(e);
c.setEmployees(employees);
em.persist(c);
c = new Company();
c.setName("Sun Microsystems");
employees = new ArrayList<Employee>();
e = new Employee();
e.setName("Rima Patel"); e.setSex('F'); employees.add(e);
e = new Employee();
e.setName("James Gosling"); e.setSex('M'); employees.add(e);
c.setEmployees(employees);
em.persist(c);
```

Relation – 1:N bidir – “from”

```
...
@Entity(name=”CompanyOMUni”)
public class Company implements Serializable {
    private int id;
    private String name;
    private Collection<Employee> employees;
    ...
    // other getters and setters go here
    // including the Id
    ...
    @OneToMany(cascade={CascadeType.ALL},fetch=FetchType.EAGER,
    mappedBy=”company”)
        public Collection<Employee> getEmployees() {
        return employees;
    }
    public void setEmployees(Collection<Employee> employees) {
        this.employees = employees;
    }
}
```

Relation – 1:N bidir – “to”

```
...
@Entity(name=”EmployeeOMUni”)
public class Employee implements Serializable {
    private int id;
    private String name;
    private char sex;
    private Company company;
    ...
    // other getters and setters go here
    // including the Id
    @ManyToOne
    public Company getCompany() {
       return company;
    }
    public void setCompany(Company company) {
       this.company = company;
    }
}
```

Relation – M:N

The rules for generating a join table are:

1. The name of the join table will be the name of the owning entity, followed by an underscore (_), followed by the name of the target entity.

2. The name of the first column in the join table will be the property name, followed by an underscore, followed by the primary key name in the owner entity.

3. The name of the second column in the join table will be the property name, followed by an underscore, followed by the primary key name in the target entity.

4. The types of the columns in the join table will match the primary key types of the tables that will be referenced by it.

Relation – M:N unidir – “from”

```
...
@Entity(name=”StudentUni”)
public class Student implements Serializable {
    private int id;
    private String name;
    private Collection<Course> courses = new ArrayList<Course>();
    public Student() { id = (int)System.nanoTime(); }
    @Id
    public int getId() { return id; }
    ...
    //other setters and getters go here
    ...
    @ManyToMany(cascade={CascadeType.ALL},fetch=FetchType.EAGER)
    @JoinTable(name=”STUDENTUNI_COURSEUNI”)
    public Collection<Course> getCourses() {
        return courses;
    }
    public void setCourses(Collection<Course> courses) {
        this.courses = courses;
    }
}
```

Relation – M:N unidir – “to”

```
...
@Entity(name=”CourseUni”)
public class Course implements Serializable {
private int id;
private String courseName;
private Collection<Student> students = new ArrayList<Student>();
...
//setters and getters go here
...
}
```

Relation – M:N bidir – “from”

```
...
@Entity(name=”StudentUni”)
public class Student implements Serializable {
    private int id;
    private String name;
    private Collection<Course> courses = new ArrayList<Course>();
    public Student() { id = (int)System.nanoTime(); }
    @Id
    public int getId() { return id; }
    ...
    //other setters and getters go here
    ...
    @ManyToMany(cascade={CascadeType.ALL},fetch=FetchType.EAGER)
    @JoinTable(name=”STUDENTUNI_COURSEUNI”)
    public Collection<Course> getCourses() {
        return courses;
    }
    public void setCourses(Collection<Course> courses) {
        this.courses = courses;
    }
}
```

Relation – M:N bidir – “to”

```
...
@Entity(name=”CourseBid”)
public class Course implements Serializable {
private int id;
private String courseName;
private Collection<Student> students = new ArrayList<Student>();
    ...
    //getters and setters go here
    ...
    @ManyToMany(cascade={CascadeType.ALL},
    fetch=FetchType.EAGER,mappedBy=”courses”)
        public Collection<Student> getStudents() {
        return students;
    }
    public void setStudents(Collection<Student> students) {
        this.students = students;
    }
}
```