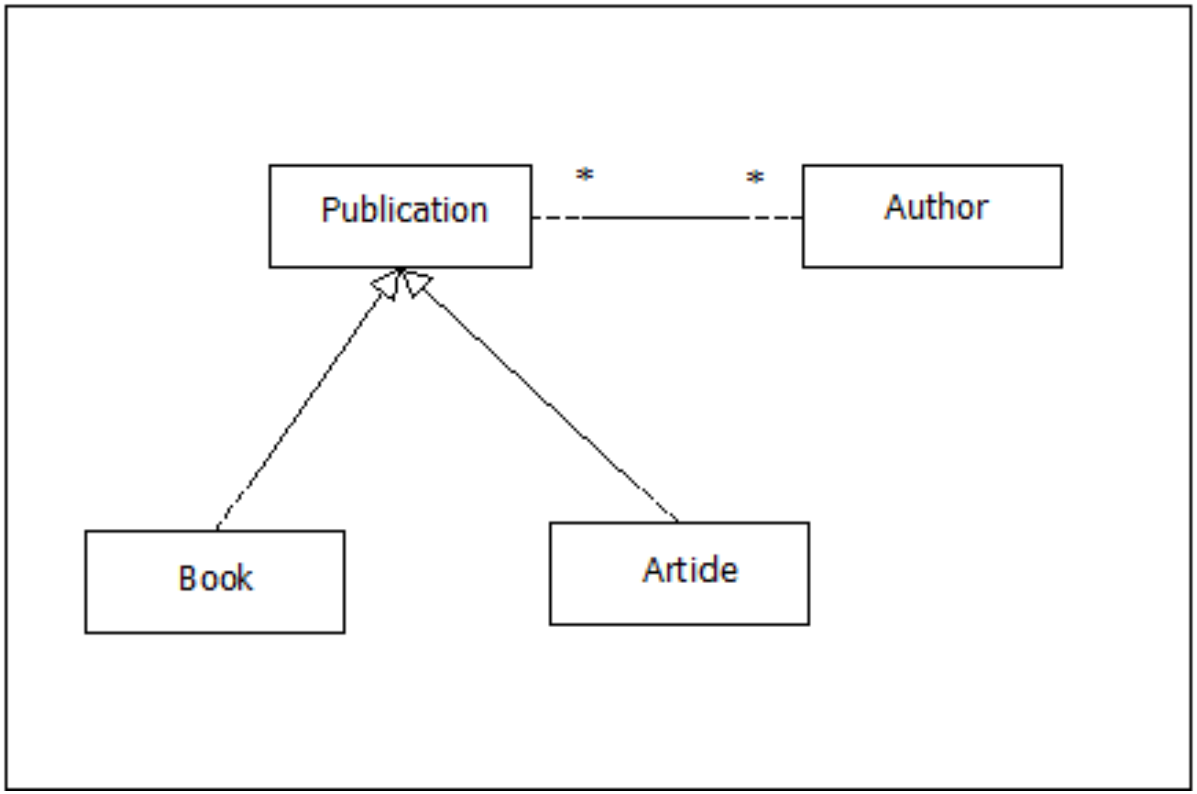
Object – Oriented Technology

Chapter 6: Object Databases

Suad Alagić

Springer 2016

SAMPLE SCHEMA



SAMPLE SCHEMA

```
class Author (extent Authors) {
  attribute string name;
  relationship set<Publication> publicationOf
        inverse Publication::authoredBy;
  string getName();
}

class Publication (extent Publications) {
  attribute string title;
  attribute integer year;
  relationship list<Author> authoredBy
        inverse Author::publicationOf;
}
```

SCHEMA with INHERITANCE

class Article **extends** Publication (**extent** Articles) {
  attribute string journal;
}

class Book **extends** Publication (**extent** Books) {
  attribute double price;
}

SAMPLE QUERY

select p.title
from Publications p
where p.year $>$ 1995
and count(p.authoredBy) $>$ 1

SAMPLE QUERY

select p.title
from Authors a, a.publicationOf p
where a.name = "Tony Hoare"

SAMPLE QUERY

select distinct a.name
from Authors a

SAMPLE QUERY

select p.title
from Publications p
order by p.year desc

NESTED QUERY

```
select distinct a.name
from (select p
        from Authors a, a.publicationOf p
        where a.name = "Tony Hoare") p
where p.authoredBy a
```

NESTED QUERY

```
select p.title
from Articles p,
        (select b.year
        from Books b
        where b.title= "Inferno") y
where p.year in y
```

QUERIES with QUANTIFIERS

select a.name
from Authors a
where exists b **in** Books:
b.price $<$ 200 **and** b **in** a.publicationOf

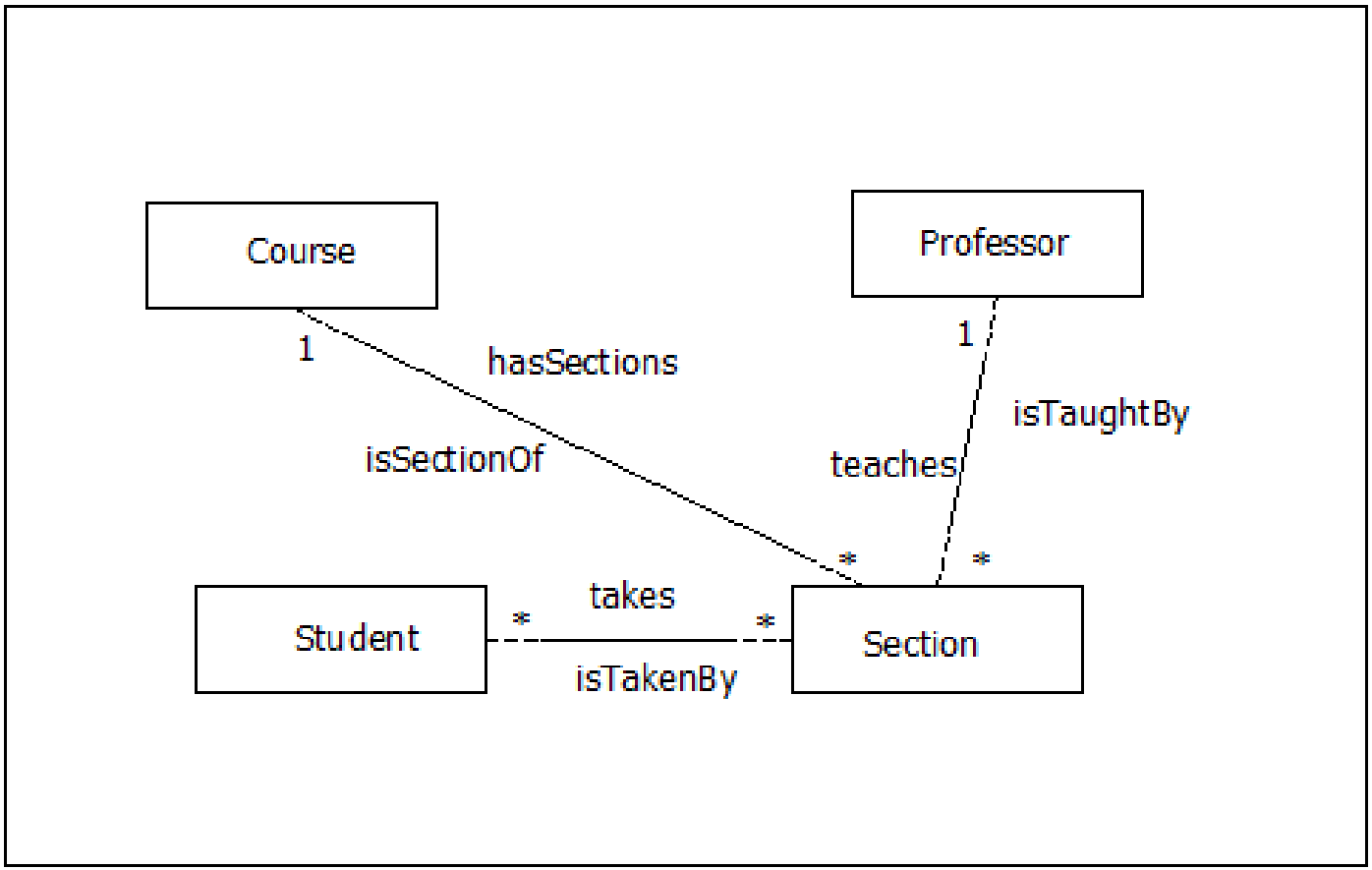
QUERIES with QUANTIFIERS

select a.name
from Authors a
where forAll p **in** a.publicationOf:
p.year $\leq$ 2000

DATABASE COLLECTIONS

```
public interface DCollection extends java.util.Collection {
  public Object selectElement(String predicate)
        throws QueryInvalidException;
  public java.util.Iterator select(String predicate)
        throws QueryInvalidException;
  public DCollection query(String predicate)
        throws QueryInvalidException;
  public DCollection existsElement(String predicate)
        throws QueryInvalidException;
}
```

SAMPLE SCHEMA



OQL QUERY

```
DCollection specialStudents;
specialStudents = Students.query(
        "exists s in this.takes: s.isTaughtBy.name="Turing");
```

SAMPLE SCHEMA

```
class Employee {
  public Employee(String name, short id,
         Date hiringDate, float salary){...}
  public String getName(){...}
  public short getId(){...}
  public Date getHiringDate(){...}
  public void setSalary(float s){...}
  public float getSalary(){...}
  private String name;
  private short id;
  private float salary;
  private Date hiringDate;
}

DCollection employees;
```

SAMPLE QUERY

```
DCollection employees, wellPaidEmployees;
...
wellPaidEmployees = employees.query("this.getSalary() ≥ 50000");
```

OQL Query CLASS

```
public interface OQLQuery {
  public void create(String query)
        throws QueryInvalidException;
  public void bind(Object parameter)
        throws QueryParameterCountInvalidException,
        QueryParameterTypeInvalidException;
  public Object execute()
        throws QueryException;
}
```

Java OQL QUERY

```
DCollection specialProfessors;
Double x;
OQLQuery query;
query=Impl.newOQLQuery();
query.create(
        "select p
        from p in Professors
        where p.salary > $1 and p in $2");
x=new Double(50000.00);
query.bind(x); query.bind(specialProfessors);
DBag selectedProfs = (DBag)query.execute();
```

LINQ: ENUMERATIONS

System.Collections.IEnumerator
System.Collections.Generic.IEnumerator<T>

LINQ: ENUMERATOR CLASS

```
class EnumeratorClass
  // implements IEnumerator or IEnumerator<T>
{
  public IteratorVariableType Current { get {...} }
  public bool moveNext() {...}
}
```

LINQ: ENUMERABLE INTERFACE

System.Collections.IEnumerable
System.Collections.Generic.IEnumerable<T>

LINQ: ENUMERABLE CLASS

```
class EnumerableClass
  // implements IEnumerable or IEnumerable<T>
{
  public Enumerator getEnumerator() {...}
}
```

LINQ QUERY

```
IEnumerable<String> query =
  from e in employees
  where e.salary() ≥ 100,000
  orderby e.salary()
  select e.name();

foreach (String name in query) Console.WriteLine (name);
```

LINQ QUERY

```
IEnumerable<String> query = employees
  .Where (e ⇒ e.salary() ≥ 100,000)
  .OrderBy (e ⇒ e.salary())
  .Select (e ⇒ e.name());
```

LINQ QUERY

```
class EmpSalary {
  String name;
  float salary;
  // . . .
}
IEnumerable<EmpSalary> =
  from e in employees
  select new EmpSalary
        { name = e.name();
        salary= e.salary() }
  where e.salary() ≥ 100,000;
```

NESTED QUERY

```
IEnumerable<String> =
  from e in employees
  where e.salary() ≥
            Max(from m in employees
            where e.manager()=m
            select m.salary())
  select e.name();
```

SQL TABLE

```
create table Employee
(
  ID int not null primary key,
  Name varchar(30)
)
```

INTERFACING TABLES

```
[Table] public class Employee
{
  [Column(IsPrimaryKey=true)]
  public int ID;
  [Column]
  public String Name;
}
```

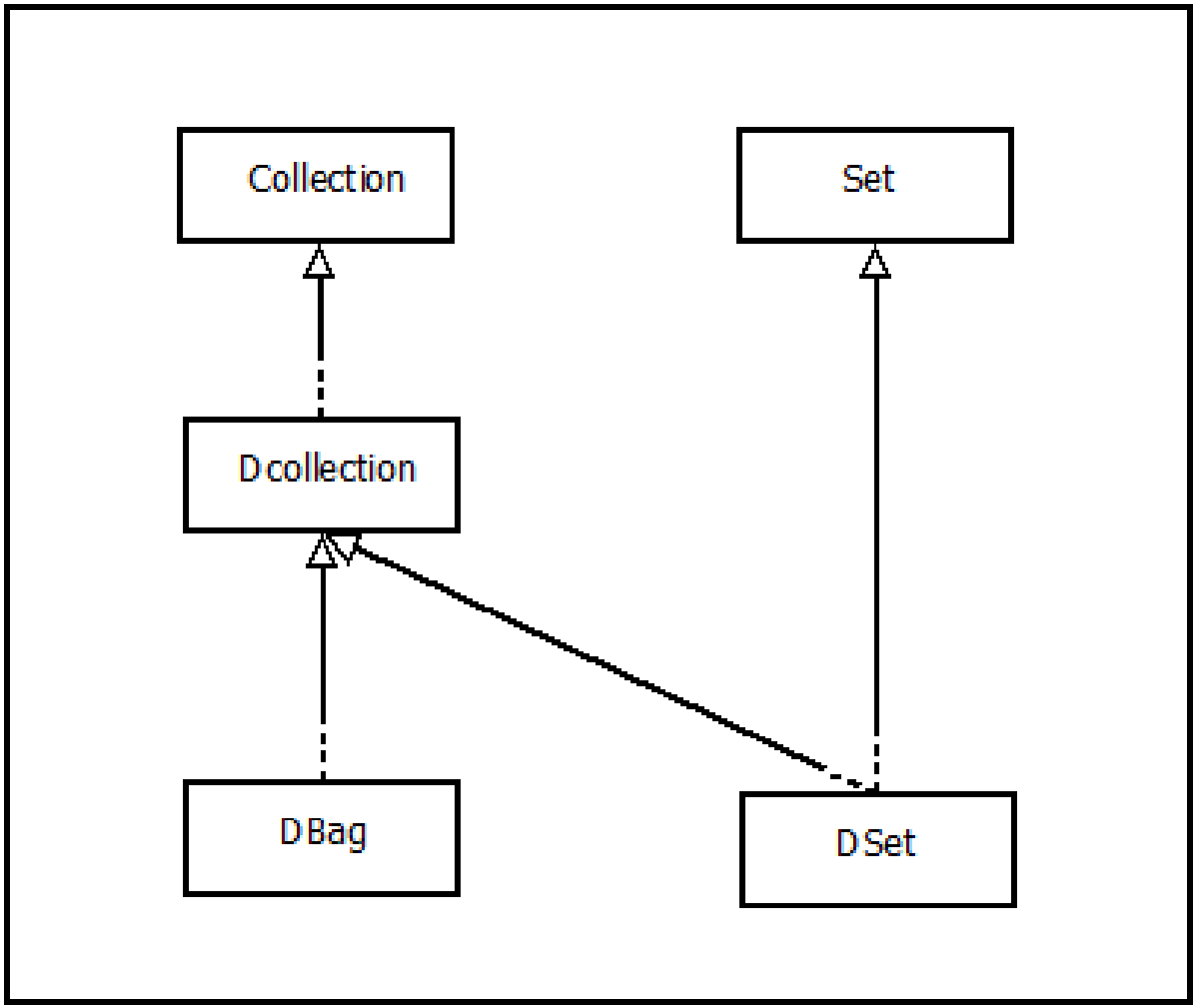
```
DataContext dataContext = new DataContext ("connection string");
Table<Employee> employees =
        dataContext.getTable <Employee>();
IQueryable<String> query =
  from c in employees
  where c.Name.First="A"
  orderby c.Name.Length
  select c.Name;


foreach (String name in query) Console.WriteLine (name);
```

DATABASE COLLECTIONS

public interface DCollection **extends** java.util.Collection {. . .}
public interface DSet **extends** DCollection, java.util.Set{. . .}
public interface DBag **extends** DCollection {. . .}

DATABASE COLLECTIONS



DATABASE CLASS

```
public interface Database {
// access modes
  public void open(String name, int accessMode)
        throws ODMGException;
  public void close()
        throws ODMGException;
  public void bind(Object object, String name)
        throws ObjectNameNotUniqueException;
  public Object lookup(String name)
        throws ObjectNameNotFoundException;
// other methods
}
```

SAMPLE CLASS

```
public class WellPaid {
  public WellPaid(String name, float salary){...}
  public String getName(){...}
  public float getSalary(){...}
  private String name;
  private float salary;
};
```

SAMPLE QUERY

```
DBag selectedEmployees = new DBag();
// open the employee database
OQLQuery aQuery = Impl.newOQLQuery(
        "select wellPaid(e.getName(), e.getSalary())
        from e in employees
        where e.getSalary() ≥ $1
        and e.getHiringDate().after($2)");
aQuery.bind(new Float(50000));
aQuery.bind(NewYears);
selectedEmployees = (DBag)query.execute();
```

CREATING QUERY OBJECTS

```
Database d = Database.open("employeeDatabase",
                openReadWrite);
OQLQuery aQuery = Impl.newOQLQuery("
        select wellPaid(e.getName(),e.getSalary())
        from e in employees
        where e.getSalary() ≥ $1
        and e.getHiringDate().after($2)");
d.bind(aQuery,"sampleQuery");
d.close();
```

ACCESSING QUERY OBJECTS

```
Database d = Database.open("employeeDatabase",
                    openReadWrite);
OQLQuery aQuery;
aQuery = (OQLQuery) d.lookup("sampleQuery");
aQuery.bind(new Float(50000));
aQuery.bind(NewYears);
DBag selectedEmployees = (DBag)query.execute();
d.bind(selectedEmployees, "wellPaidEmployees");
d.close();
```

TRANSACTION CLASS

```
public interface Transaction {
  public void begin();
  public void commit();
  public void abort();
  public void checkpoint();
// other methods
}
```

SAMPLE TRANSACTION

```
Database db = new Database("server parameters");
  db.connect();
  db.open("database name");
  Transaction Tx = new Transaction();
  Tx.begin();
        DCollection people = new DCollection();
        Database.bind(people, "People");
  Tx.commit();
  db.close();
  db.disconnect();
```

SAMPLE TRANSACTION

```
Transaction Tx = new Transaction();
Tx.begin();
  DCollection people =
        (DCollection) Database.lookup("People");
  Person p= new Person("John", 35);
  people.add(p);
Tx.commit();
```

SAMPLE TRANSACTION

```
Transaction Tx = new Transaction();
Tx.begin();
  Person john = new Person("John", 30);
  Person mary = new Person("Mary", 28);
  john.assignSpouse(mary);
  mary.assignSpouse(john);
DCollection people = new DCollection();
try {Database.bind(people, "People");}
  catch (DbException e) { handle exception };
  people.add(john);
  people.add(mary);
Tx.commit();
```

ACESSING DATABASE

```
IObjectContainer db = Db4oFactory.OpenFile(FileName);
try {
  access db4o
  }
finally
  { db.close(); }
```

SAMPLE CLASS

```
public class Pilot
{ String name;
  int points;
  public Pilot(String pName, int pPoints)
  { name = pName; points = pPoints; }
  public String Name {
  { get {return name; }
  { void set {name = value;}
  }
  public int Points {
  { get {return points; }
  {void set{points = value;}
  }
  public void addPoints(int Ppoints)
  { points += Ppoints; }
  }
```

STORING OBJECTS

```
Pilot pilot = new Pilot("Mark Sellinger", 100);
db.store(pilot);
```

ACCESSING OBJECTS

```
Pilot protoObj = new Pilot("Mark Sellinger", 0);
IObjectSet result = db.get(protoObj);
```

SELECTING OBJECTS

```
Pilot protoObj = new Pilot(null, 100);
IObjectSet result = db.get(protoObj);
```

```
Pilot protoObj = new Pilot(null, 0);
IObjectSet result = db.get(protoObj);
```

UPDATING OBJECTS

```
IObjectSet result = db.get(new Pilot("Mark Sellinger", 0));
Pilot found = (Pilot) result.next();
found.addPoints(50);
db.set(found);
```

DELETING OBJECTS

```
IObjectSet result = db.get(new Pilot("Mark Sellinger", 0));
Pilot found = (Pilot)result.next();
db.delete(found);
```

NATIVE QUERIES

```
List <Pilot> result = db.query(new Predicate<Pilot>() {
  public boolean match(Pilot pilot) {
  return (pilot.Points ≥ 99
              ∧ pilot.Points ≤ 199)
         ∨ pilot.Name.equals("Mark Sellinger");
  }
});
```

```
public class Aircraft
{ String model;
  Pilot pilot;

  public Aircraft(String aModel)
  { model = aModel; pilot = null; }

  public Pilot Pilot {
        get { return pilot;}
        set { pilot = value;}
  }
public String Model
{ get { return model; }
  }
}
```

COMMITING COMPLEX OBJECTS

```
Pilot pilot = new Pilot("Mark Sellinger", 99);
Aircraft plane = new Aircraft("Boeing 777");
plane.Pilot = pilot;
db.set(plane);
db.commit();
```

ROLLBACK

```
Pilot pilot = new Pilot("Mark Sellinger", 100);
Aircraft plane = new Aircraft("Boeing 777");
plane.Pilot = pilot;
db.set(plane);
db.rollBack();
```

SAMPLE APPLICATION

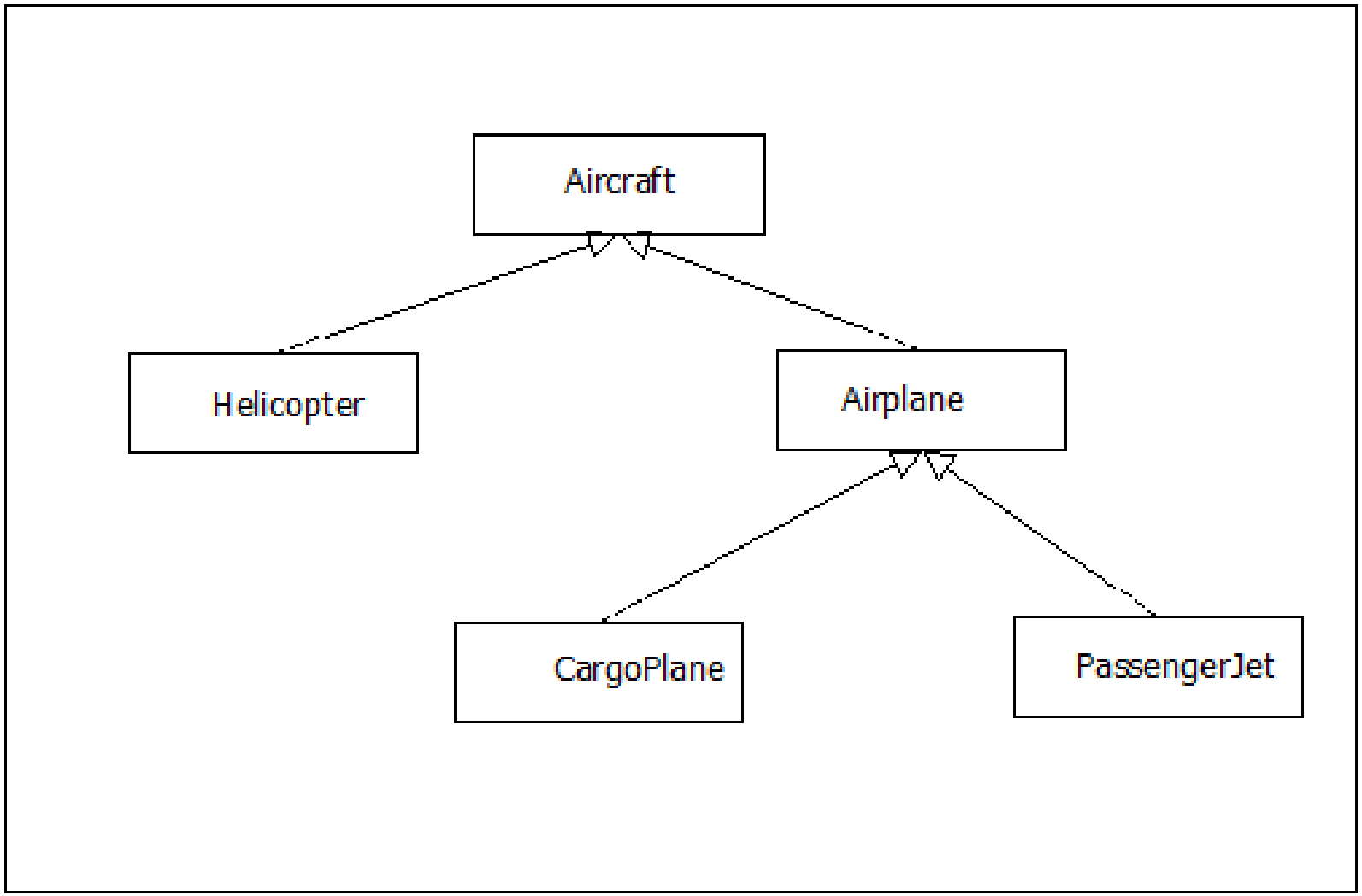


Figure: Aircraft hierarchy

```
public abstract class Aircraft {
  protected String aircraftId;
  // constructor and methods
}
```

```
public class Airplane extends Aircraft {
  protected List<AirplaneEngine> engines;
  // constructor and methods
}
```

```
public class CargoPlane extends Airplane {
  private int storageCapacity;
  // constructor and methods
}
```

```
public class PassengerJet extends Airplane {
  private int maxPassengers;
  // constructor and methods
}
```

STORING OBJECTS

```
private void saveObject(Object object) {
  try {db.store(object);
        db.commit();
        }
  catch (Db4oIOException | DatabaseClosedException |
        DatabaseReadOnlyException e)
        {exception handling }
  }
```

```
public ObjectSet<Object> findAirplaneById(String id)
{
  Airplane plane = new Airplane(id, null);
 try { return db.queryByExample(plane);
        } catch (Db4oIOException | DatabaseClosedException e)
        { exception handling;}
  return null;
}
```

```
public List<Aircraft> selectAirplanes() {
  List<Aircraft> matches =
  db.query(new Predicate<Aircraft>() {
  public boolean match(Aircraft aircraft) {
        if (aircraft instanceof Airplane) {
        Airplane airplane = (Airplane) aircraft;}
        if (airplane.getEngines().size() ≥ 500)
        return true;
        else return false;
  });
  return matches;
}
```