

# Software Engineering: Specification, Implementation, Verification

## Chapter 4: Mapping Models to Code

Suad Alagić

*Springer 2017*

- Specifying interfaces
- From interfaces to classes
- Model and code management
- Interplay of inheritance and constraints
- Models with complex constraints

# Results of analysis

- Use cases
- Entity types
- Association relationships
- Inheritance relationships
- Pre- and post-conditions (informally)

# Declarative specification

- Specifying interfaces
- Specifying relationships
- Specifying pre- and post conditions in OCL
- Declarative software specification

# Specifying interfaces

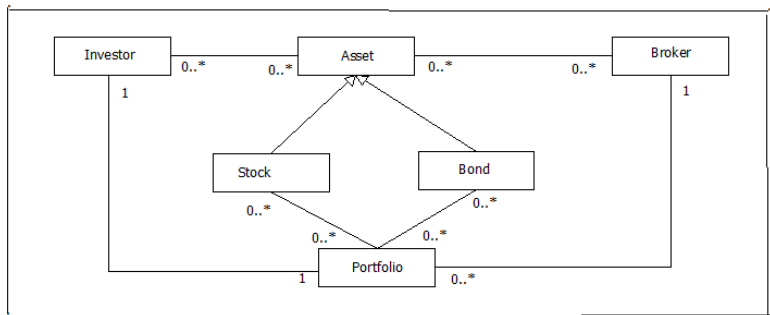


Figure: Investment management entity types and their relationships

# Specifying interfaces

```
interface IAsset {  
    float getPrice();  
    boolean priceOk();  
    Collection < Broker > getBrokers();  
    Collection< Portfolio > getPortfolios();  
}
```

```
interface IBroker {  
    Collection<Asset> getAllAssets();  
    Collection <Portfolio> getMyPortfolios();  
}
```

# Specifying interfaces

```
interface IPortfolio {  
    Collection<Asset> getPortfolioAssets();  
    Broker getBroker();  
    Investor getInvestor();  
}
```

```
interface InvestorI {  
    Portfolio getMyPortfolio();  
    Broker getMyBroker();  
    Collection<Asset> getAllAssets();  
    void buyAsset(Asset a);  
    void sellAsset(Asset a);  
}
```

# Specifying interfaces

**context** InvestorI:: buyAsset(Asset a):  
**pre not** (self.getMyPortfolio().getPortfolioAssets() – > includes(a))  
**pre** a.priceOk();  
**post** self.getMyPortfolio().getPortfolioAssets() – > includes(a)

**context** InvestorI:: sellAsset(Asset a):  
**pre** self.getMyPortfolio().getPortfolioAssets() – > includes(a)  
**post not** (self.getMyPortfolio().getPortfolioAssets() – > includes(a))



# Specifying interfaces

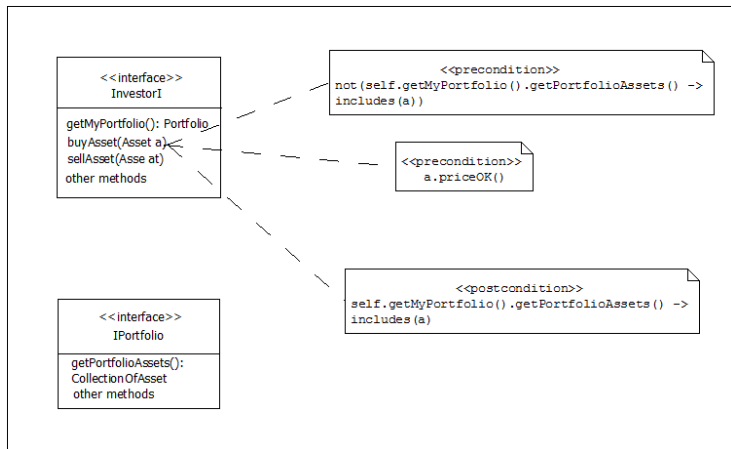


Figure: Assertions for buying an asset

# Specifying interfaces

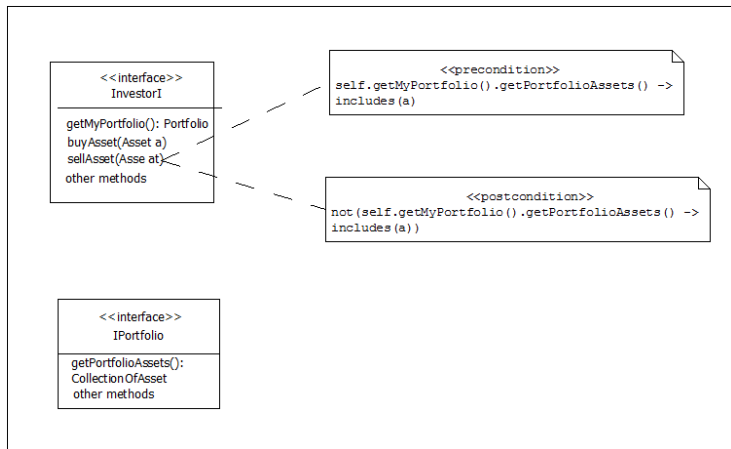


Figure: Assertions for selling an asset

# Entity diagram

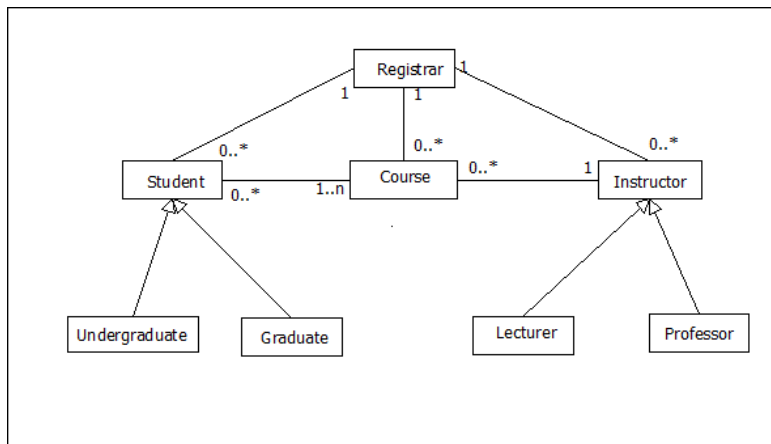


Figure: Course management entities and their relationships

```
interface IRegistrar {  
    Collection<Course> getAllCourses();  
    Collection<Instructor> getInstructors();  
    Collection<Students> getStudents();  
    Collection<Room> getRooms();  
    void scheduleCourse(Course c);  
    void deleteCourse(Course c);  
}
```

# Interface constraints

```
context IRegistrar:: scheduleCourse(Course c):  
pre not (self.getAllCourses() – > includes(c))  
pre self.getRooms() – > exists(r: Room | r.suitableFor(c))  
pre self.getInstructors() – > exists (x: InstructorI | x.suitableFor(c))  
post self.getAllCourses() – > includes(c)
```

# Interface constraints

```
context IRegistrar:: deleteCourse(Course c):  
pre self.getAllCourses() – > includes(c)  
post not (self.getAllCourses() – > includes(c))
```

# Interface constraints

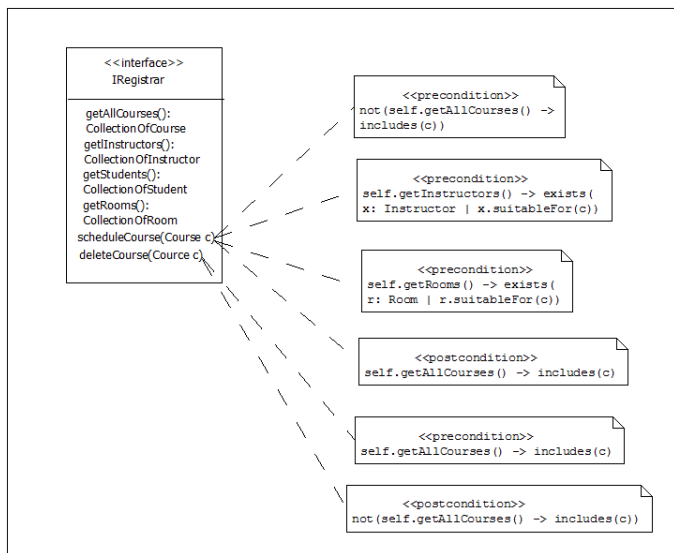


Figure: Course scheduling constraints

```
interface IStudent {  
    Collection<Course> getAllCourses();  
    Collection<Course> getMyCourses();  
    void enrollInCourse(Course c);  
    void dropCourse(Course c);  
}
```



# Interface constraints

```
context IStudent:: enrollInCourse(Course c):  
pre not (self.getMyCourses() – > includes(c))  
pre c.getPrerequisites() – > subset(self.getMyCourses())  
pre c.open()  
post self.getMyCourses() – > includes(c)
```

# Interface constraints

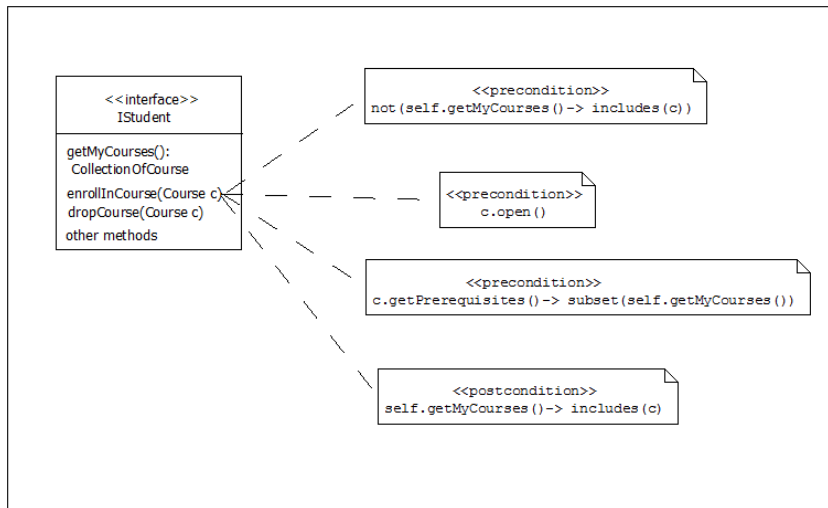


Figure: Assertions for enrolling in a course

# Interface constraints

```
context IStudent:: dropCourse(Course c):  
pre self.getMyCourses() – > includes(c)  
post not (self.getMyCourses() – > includes(c))
```

# Interface constraints

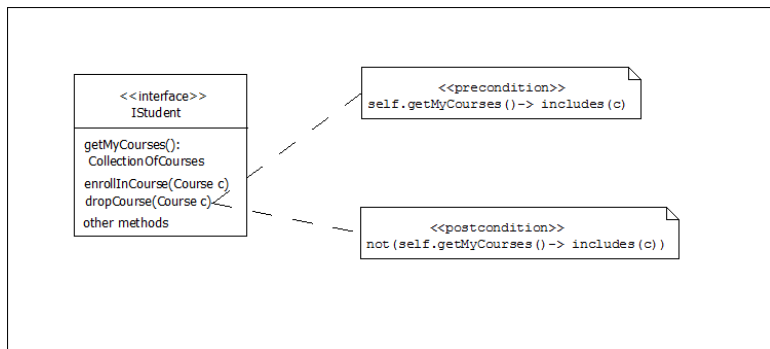


Figure: Assertions for dropping a course

```
interface ICourse {  
    boolean open();  
    InstructorI getInstructor();  
    Collection<Course> getPrerequisites();  
    Collection<Student> getMyStudents();  
    Time getTime();  
    Room getClassRoom();  
}
```

```
interface InstructorI {  
    Collection<Course> getMyCourses();  
    boolean suitableFor(Course c);  
}
```

```
interface IRoom {  
    boolean suitableFor(Course c);  
    int maxCapacity();  
}
```

# Specifying classes

- Implementing associations
- Persistent collections
- Implementing interfaces as classes.
- Producing code for methods
- Code management

# Specifying classes

```
Collection<Asset> assets;  
Collection<Portfolio> portfolios;  
Collection<Investor> investors;  
Collection< Broker> brokers;
```



# Specifying classes

```
class Asset implements IAsset {  
    private float price;  
    public getPrice() {  
        return this.price;  
    }  
    public boolean priceOk() {  
        return this.price <= getAcceptableValue();  
    }  
    public void setPrice(float price) {  
        this.price = price;  
    }  
}
```

# Specifying classes

```
public Collection<Broker> getBrokers() {  
  return (from b in brokers  
    where this in b.getMyPortfolios().getPortfolioAssets()  
    select b);  
}  
// method getPortfolios  
}
```

# Specifying classes

```
Collection<Broker> getBrokers() {  
    Collection<Broker> result;  
    for (Broker b: brokers)  
        if (b.getMyPortfolios().getPortfolioAssets().contains(this))  
            result.include(b);  
    return result;  
}
```

# Specifying classes

```
class Portfolio implements IPortfolio {  
    private Investor owner;  
    private Broker manager;  
    private Collection< Stock > stocks;  
    private Collection< Bond > bonds;  
    public Broker getBroker() {  
        return this.manager; }  
    public Investor getInvestor() {  
        return this.owner; }  
    public Collection< Stock> getStocks() {  
        return stocks;  
    }  
    public Collection<Bond> getBonds() {  
        return bonds;  
    }  
}
```

# Specifying classes

```
public Collection< Asset > getPortfolioAssets() {  
  return (from a in assets  
    where (Stock)a in this.getStocks()  
    or (Bond)a in this.getBonds()  
    select a);  
}  
}
```

# Specifying classes

```
Collection<Asset> getPortfolioAssets() {  
    Collection<Asset> result;  
    for (Asset a: assets)  
        if stocks.contains((Stock)a) or  
        bonds.contains((Bond)a )  
            result.include(a);  
    return result;  
}
```

# Specifying classes

```
class Investor implements InvestorI {  
    private Portfolio myPortfolio;  
    public Broker getMyBroker() {  
        return this.getMyPortfolio().getBroker();  
    }  
    public Portfolio getMyPortfolio() {  
        return this.myPortfolio; }  
    public Collection<Asset> getAllAssets() {  
        return assets; }  
}
```

# Specifying classes

```
public void buyAsset (Asset a) {  
    // code  
}  
public void sellAsset (Asset a) {  
    // code  
}  
// ...  
}
```



# Specifying classes

```
class Broker implements IBroker {  
  private String brokerId;  
  private name;  
  public String getBrokerId() {  
    return this.brokerId; }  
  public String getBrokerName() {  
    return this.name; }  
  public Collection<Asset> getAllAssets() {  
    return assets; }  
  public Collection<Portfolio> getMyPortfolios() {  
    return (from p in portfolios  
      where p.getBroker()= this  
      select p);  
  }  
  //...  
}
```

# Specifying classes

```
Collection<Portfolio> getMyPortfolios(){  
    Collection<Portfolio> result;  
    for (Portfolio p: portfolios)  
        if (p.getBroker() = this)  
            result.include(p)  
    return result;  
}
```

# Specifying classes

```
Collection<Course> courses;  
Collection<Student> students;  
Collection<Instructor> instructors;  
Collection<Room> classrooms;  
Registrar r;
```

# Specifying classes

```
class Course implements ICourse {  
    private String courseId;  
    private String name;  
    private Instructor taughtBy;  
    private Room classroom;  
    private Time schedule;  
    public String getCourseId() {  
        return this.courseId; }  
    public String getCourseName() {  
        return this.name; }  
    public String getInstructor() {  
        return this.taughtBy; }  
    public Room getRoom() {  
        return this.classRoom; }  
    public Time getTime() {  
        return this.schedule; }  
}
```

# Specifying classes

```
public boolean open() {  
    return this.getMyStudents.size() <  
        classRoom.getMaxCapacity();  
}  
public Collection<Student> getMyStudents() {  
    return (from s in students  
        where this in s.getMyCourses( )  
        select s);  
}  
}
```

# Specifying classes

```
Collection<Student> getMyStudents() {  
    private Collection<Student> result;  
    for (Student s: students)  
        if (s.getMyCourses().contains(this))  
            result.include(s)  
}
```

# Specifying classes

```
class Student implements IStudent {  
    String studentId;  
    String name;  
    String getStudentId() {  
        return this.studentId; }  
    String geStudentName() {  
        return this.name; }  
    public Collection<Course> getAllCourses() {  
        return courses;  
    }  
    pubic Collection<Course> getMyCourses() {  
        return (from c in courses  
            where this in c.getMyStudents()  
            select c);  
    }
```

# Specifying classes

```
public void enrollInCourse(Course c) {  
    . . . }  
public void dropCourse(Course c) {  
    . . . }  
// . . .  
}
```



# Specifying classes

```
Collection<Course> getMyCourses() {  
    Collection<Course> result;  
    for (Course c: courses)  
        if (c.getMyStudents().contains(this))  
            result.include(c);  
    return result;  
}
```

# Specifying classes

```
class Room implements IRoom{  
    private int maxCapacity;  
    private int getMaxCapacity() {;  
        return maxCapacity;  
    }  
    public boolean suitableFor(Course c) {  
        //code  
    }  
}
```

# Specifying classes

```
class Registrar implements IRegistrar {  
    public Collection<Course> getAllCourses() {  
        return courses;  
    }  
    public Collection<Course> getInstructors() {  
        return instructors;  
    }  
    public Collection<Course> getStudents() {  
        return students;  
    }  
    public Collection<Room> getRooms() {  
        return rooms;  
    }  
}
```

# Specifying classes

```
public void scheduleCourse(Course c) {  
    // code  
}  
public void deleteCourse(Course c) {  
    // code  
}  
// ...  
}
```

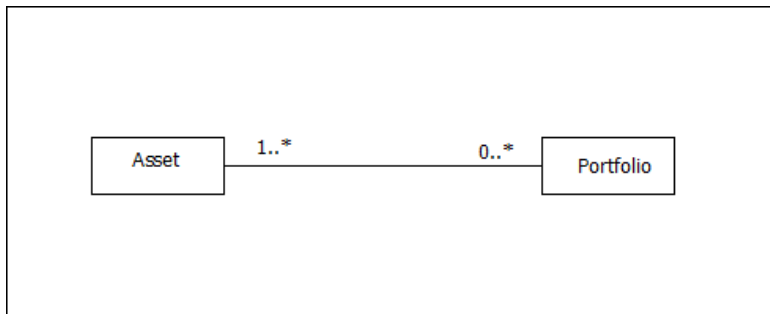


Figure: Assets and portfolios model

```
class InvestmentManagement {  
    Collection<Asset> assets;  
    Collection<Portfolio> portfolios;  
    interface IAsset  
    { . . . }  
    interface IPortfolio  
    { . . . }
```

```
class Asset implements IAsset
{ . . . }
class Portfolio implements IPortfolio
{ . . . }
// . . .
}
```

```
interface IAsset {  
    String getName();  
    String getCode();  
    Collection <Portfolio> getPortfolios();  
}
```

```
interface IPortfolio {  
    float getTotalValue();  
    Collection <Asset> getAssets();  
}
```



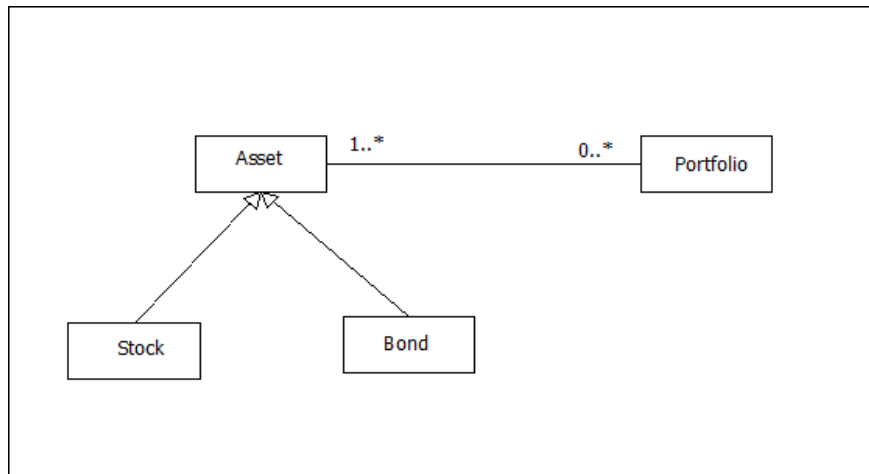


Figure: Extended assets and portfolios model

```
class ExtendedInvestmentManagement
    extends InvestmentManagement {
    interface IStock extends IAsset
    { float getShareValue();
    }
    interface IBond extends IAsset
    { float getYield();
    }
    class Stock implements IStock
    { . . . }
    class Bond implements IAsset
    { . . . }
}
```

# Model and code management

- Forward engineering
- Model transformations
- Refactoring
- Reverse engineering

# Model transformation

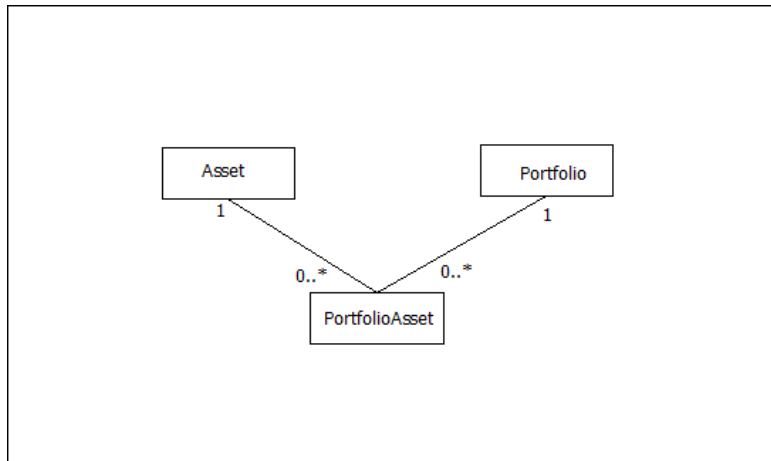


Figure: Toward relational representation

```
[Table]
class Asset {
[Column] [IsPrimaryKey=true]
    String assetId;
[Column]
    String name;
}
```

```
[Table]
class Portfolio {
[Column] [IsPrimaryKey=true]
    String portfolioId;
[Column]
    float totalValue;
}
```

```
[Table]
class PortfolioAsset {
[Column]
    String portfolioId;
[Column]
    String assetId;
}
```

```
class InvestmentManagement {  
    Table<Asset> assets;  
    Table<Portfolio> portfolios;  
    Table<PortfolioAsset> portfolioAssets;  
    interface IAsset  
    { . . . }  
    interface IPortfolio  
    { . . . }  
    interface IPortfolioAsset  
    { . . . }
```

```
class Asset implements IAsset
{ . . . }
class Portfolio implements IPortfolio
{ . . . }
class PortfolioAsset implements IPortfolioAsset
{ . . . }
}
```



```
class Asset implements IAsset {  
    String assetId;  
    String name;  
    String getAssetId() {  
        return assetId;  
    }  
    String getName() {  
        return name;  
    }  
}
```

```
Asset getAsset(String assetId)
{ return (from a in assets
      where a.assetId = assetId
      select a);
}
```

```
Table<Portfolio> getPortfolios()
{ Table<Portfolio> result;
  for (PortfolioAsset pA: portfolioAssets)
    if (pA.assetId = this.assetId)
      result.include(getPortfolio(pA.portfolioId));
  return result;
}
```

```
class Portfolio implements IPortfolio {  
    String portfolioId;  
    float totalValue;  
    String getPortfolioId() {  
        return portfolioId;  
    }  
    Float getTotalValue() {  
        return totalValue;  
    }  
    Portfolio getPortfolio(String portfolioId) {  
        if (this.portfolioId == portfolioId) return this  
            else return null;  
    }  
}
```

```
Table<Portfolio> getAssets()
{ Table<Asset> result;
  for (PortfolioAsset pA: portfolioAssets)
    if (pA.portfolioId = this.portfolioId)
      result.include(getAsset(pA.assetId));
  return result;
}
```

# Model and code transformation

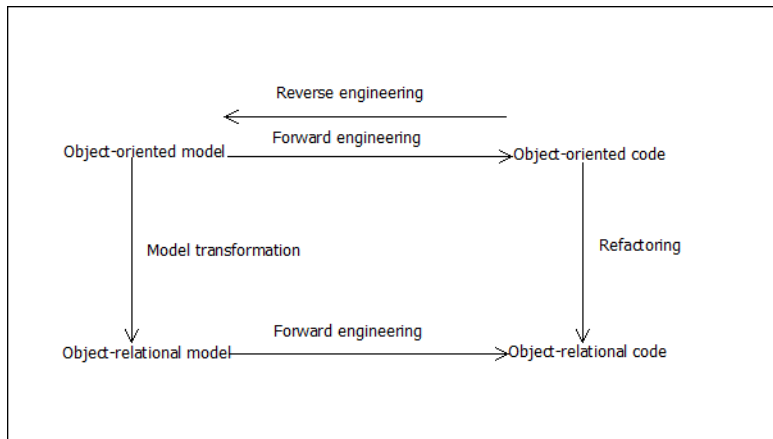


Figure: Model and code transformations

# Inheritance and constraints

- Behavioral compatibility
- Strengthening invariants
- Strengthening postconditions
- Invariance of preconditions

# Airport model

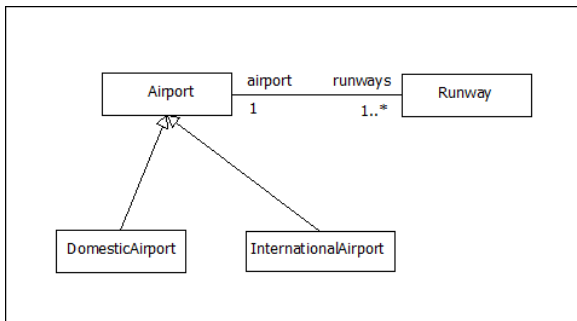


Figure: Airports and runways

```
interface IAirport {  
    int getNoOfRunways();  
    Collection<Runway> getRunways();  
    void addRunway(Runway strip);  
    void closeRunway(Runway strip);  
    // other methods  
}
```

```
interface InternationalAirportI extends IAirport {  
    // additional methods  
}
```



# Constrains for airport model

**context** Airport **inv**

self.getNoOfRunways()  $\geq 1$  **and**

self.getNoOfRunways()  $\leq 30$

# Constrains for airport model

```
context Airport:: addRunway(strip: Runway)
pre not (self.getRunways() - > exists(r: Runway | r=strip))
post self.getRunways() - > exists(r: Runway | r=strip)
```

# Constrains for airport model

```
context Airport:: closeRunway(strip: Runway)
pre self.getRunways() - > exists(r:Runway | r=strip)
post self.getNoOfRunways() >= 1
post self.getNoOfRunways() = self.getNoOfRunways()@pre -1
post self.getRunways() - > forAll (r: Runway | r <> strip)
```

# Constraints for airport model

```
context InternationalAirport inv  
  self.getNOfRunways() >= 10  
  self.getRunways() - > exists (r: Runway | r.international=true)
```

# Constrains for airport model

```
context InternationalAirport:: closeRunway(strip: Runway)
post self.getNoOfRunways()  $\geq$  10
post self.getRunways()  $- >$ 
    exists (r: Runway | r.international=true)
```