SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

12 Object Oriented Design (Ghezzi Ch. 4) DRAFT

Dr. Spencer Smith

Faculty of Engineering, McMaster University

December 15, 2017

12 Object Oriented Design (Ghezzi Ch. 4) DRAFT

- Administrative details
- OOD
- Inheritance
- Polymorphism
- Dynamic binding
- Introduction to UML

Administrative Details

TBD

Reviewing Changes

- Use GitLab to review changes between commits
- Review before committing: `git difftool`
- To better deal with changes, use a "hard wrap" at an 80 column width, even for LaTeX documents

Object Oriented Design

- One kind of module, ADT, called class
- A class exports operations (procedures) to manipulate instance objects (often called methods)
- Instance objects accessible via references
- Can have multiple instances of the class (class can be thought of as roughly corresponding to the notion of a type)

Inheritance

- Another relation between modules (in addition to USES and IS_COMPONENT_OF)
- ADTs may be organized in a hierarchy
- Class B may specialize class A
  - B inherits from A
  - Conversely, A generalizes B
- A is a superclass of B
- B is a subclass of A

Template Module Employee

Routine name	**In**	**Out**	**Except**
Employee	string, string, moneyT	Employee	
first_Name		string	
last_Name		string	
where		siteT	
salary		moneyT	
fire			
assign	siteT		

Inheritance Examples

Template Module Administrative_Staff **inherits** Employee

Routine name	**In**	**Out**	**Exception**
do_this	folderT		

Template Module Technical_Staff **inherits** Employee

Routine name	**In**	**Out**	**Exception**
get_skill		skillT	
def_skill	skillT		

Inheritance Continued

- A way of building software incrementally
- Useful for long lived applications because new features can be added without breaking the old applications
- A subclass defines a subtype
- A subtype is substitutable for the parent type
- Polymorphism - a variable referring to type A can refer to an object of type B if B is a subclass of A
- Dynamic binding - the method invoked through a reference depends on the type of the object associated with the reference at runtime
- All instances of the sub-class are instances of the super-class, so the type of the sub-class is a subtype
- All instances of Administrative_Staff and Technical_Staff are instances of Employee

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff()
emp2 = Technical_Staff()
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff()
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff() $\surd$
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff() $\surd$
emp3 = emp1 $\times$
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff() $\surd$
emp3 = emp1 $\times$
emp3 = (Technical_Staff) emp1 $\surd$

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff
emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff() $\surd$
emp3 = emp1 $\times$
emp3 = (Technical_Staff) emp1 $\surd$

Polymorphism: type of RHS must be a subtype of the LHS

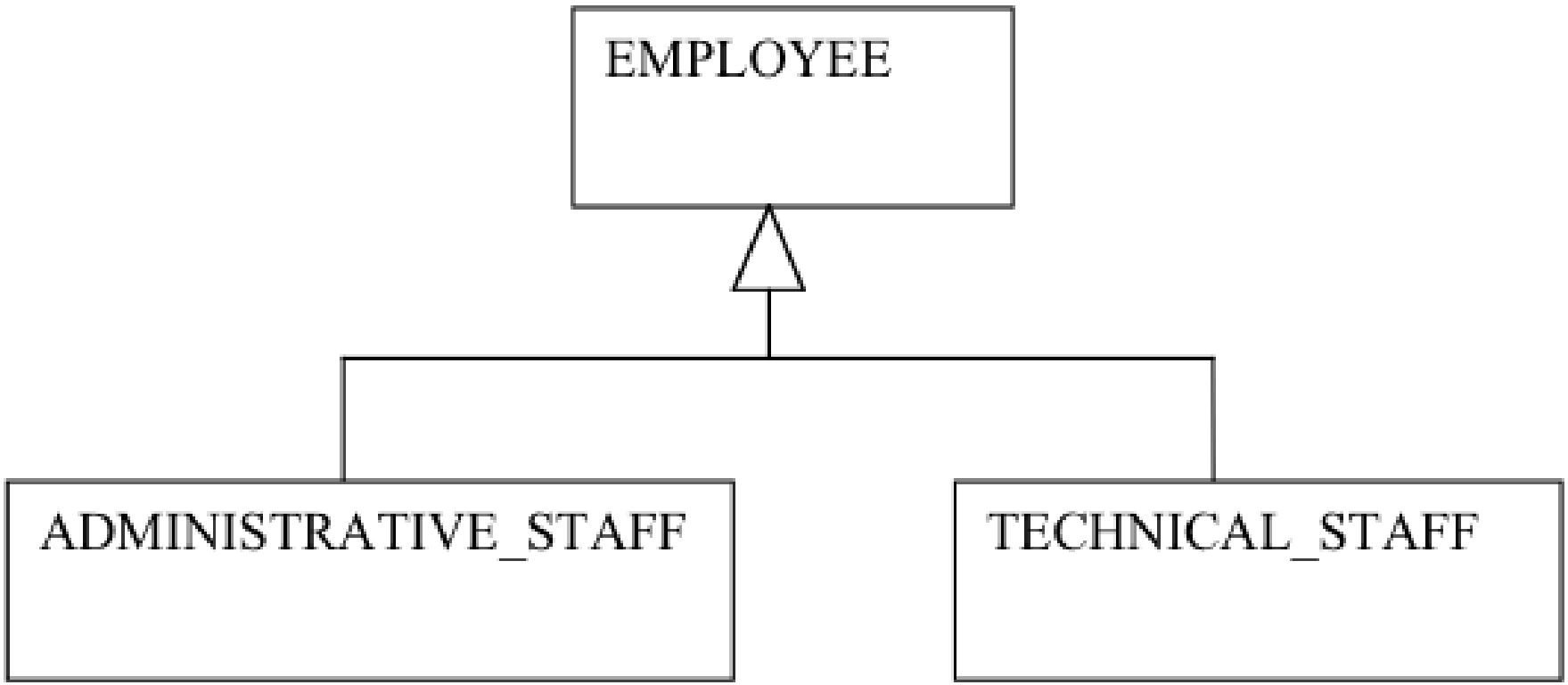
Dynamic Binding

- Many languages, like C, use static type checking
- OO languages use dynamic type checking as the default
- There is a difference between a type and a class once we know this
  - Types are known at compile time
  - The class of an object may be known only at run time

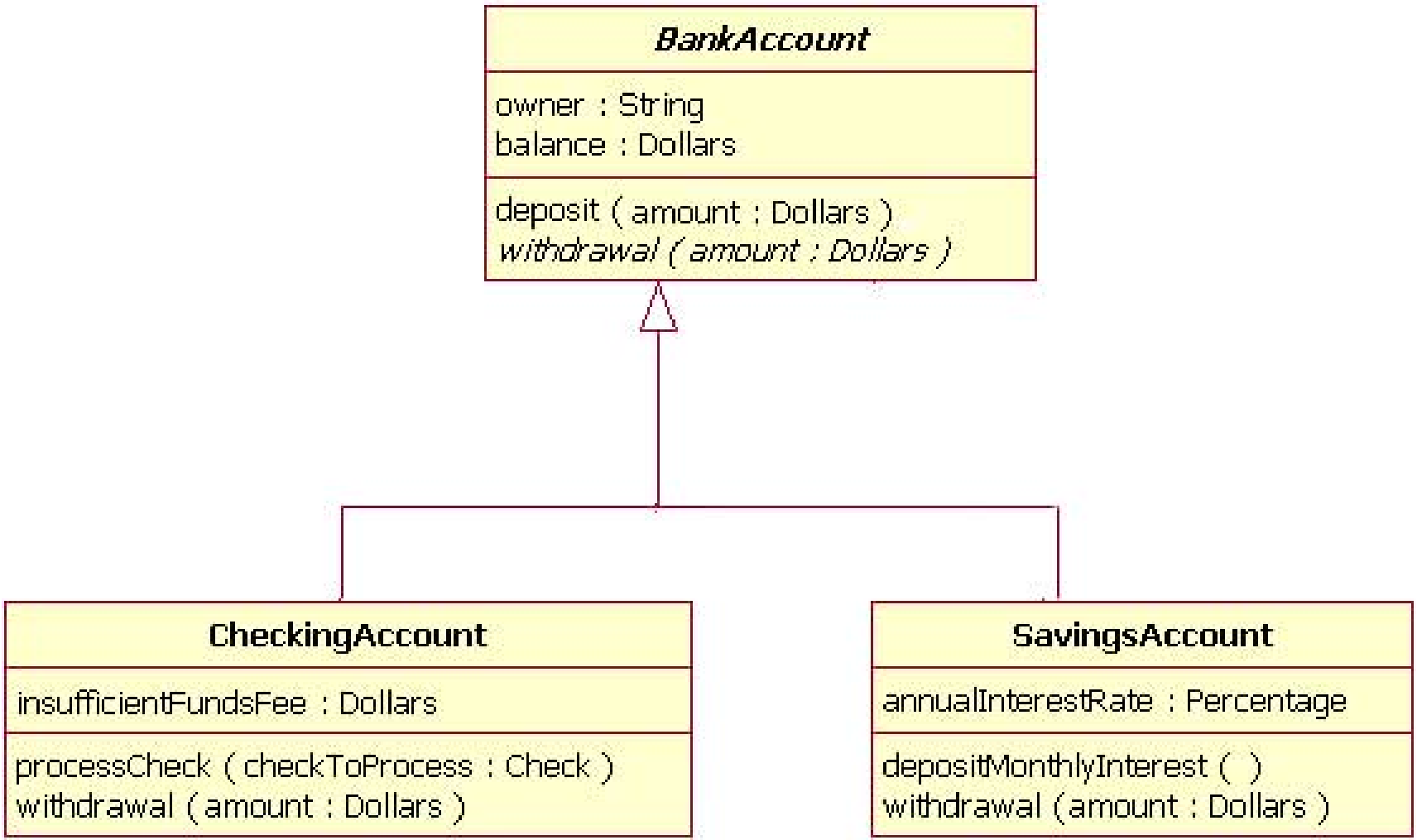
How can Inheritance be Represented?

- We start introducing the UML notation
- UML (Unified Modelling Language) is a widely adopted standard notation for representing OO designs
- We introduce the UML class diagram
- Classes are described by boxes

UML Representation of Inheritance

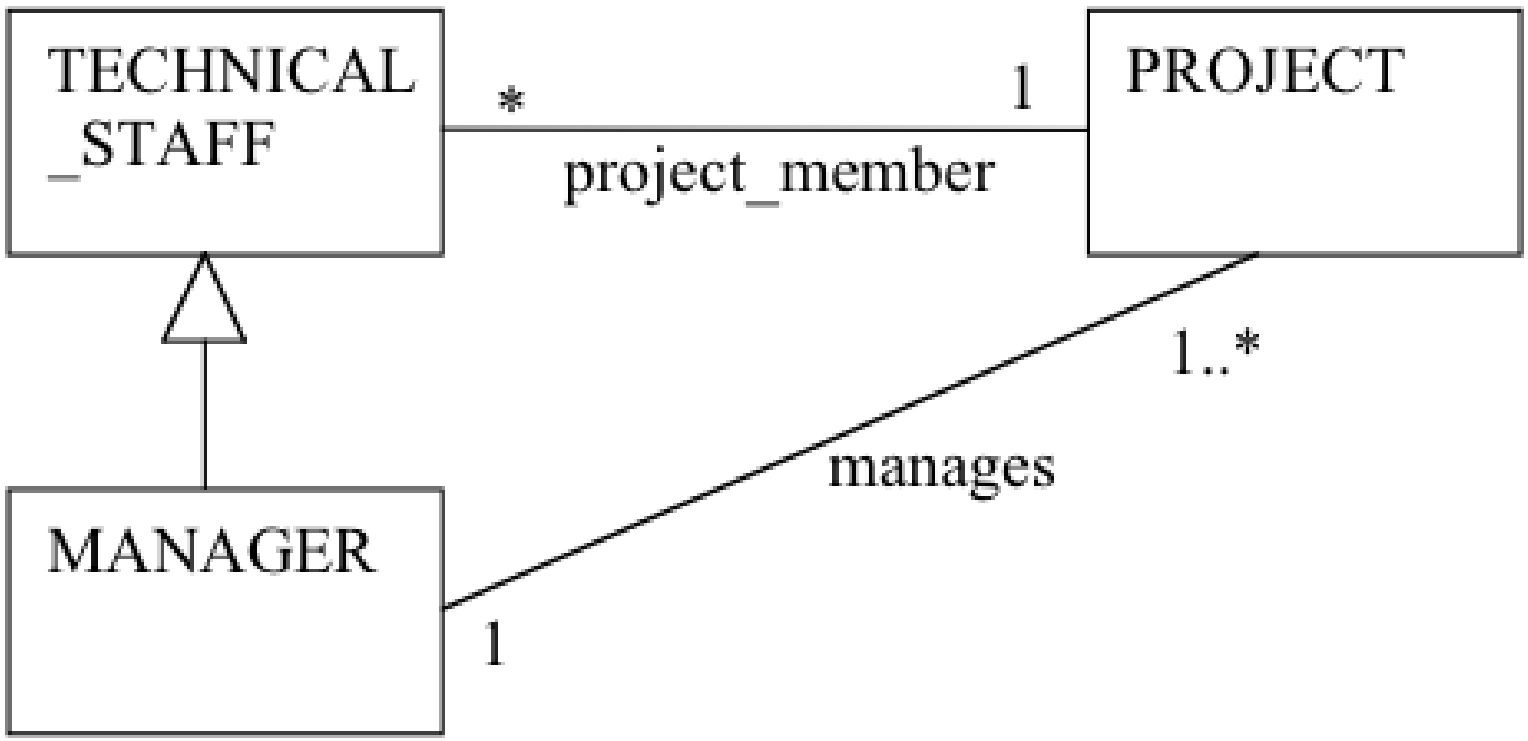


Bank Account Example

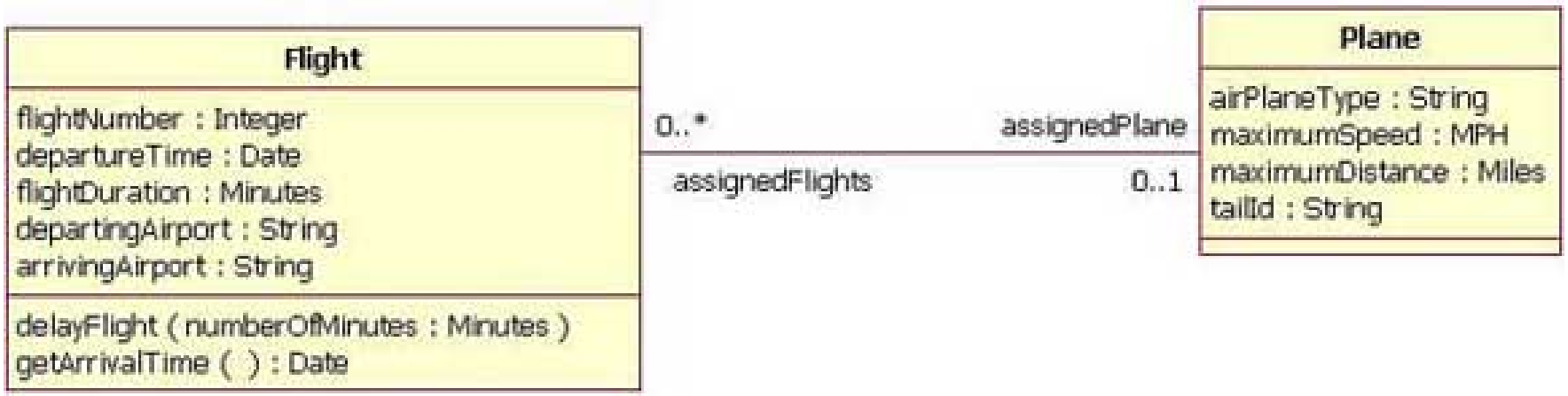


UML Associations

- Associations are relations that the implementation is required to support
- Can have multiplicity constraints

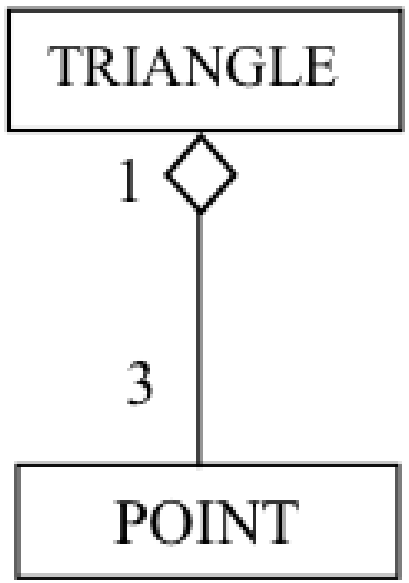


Flight Example



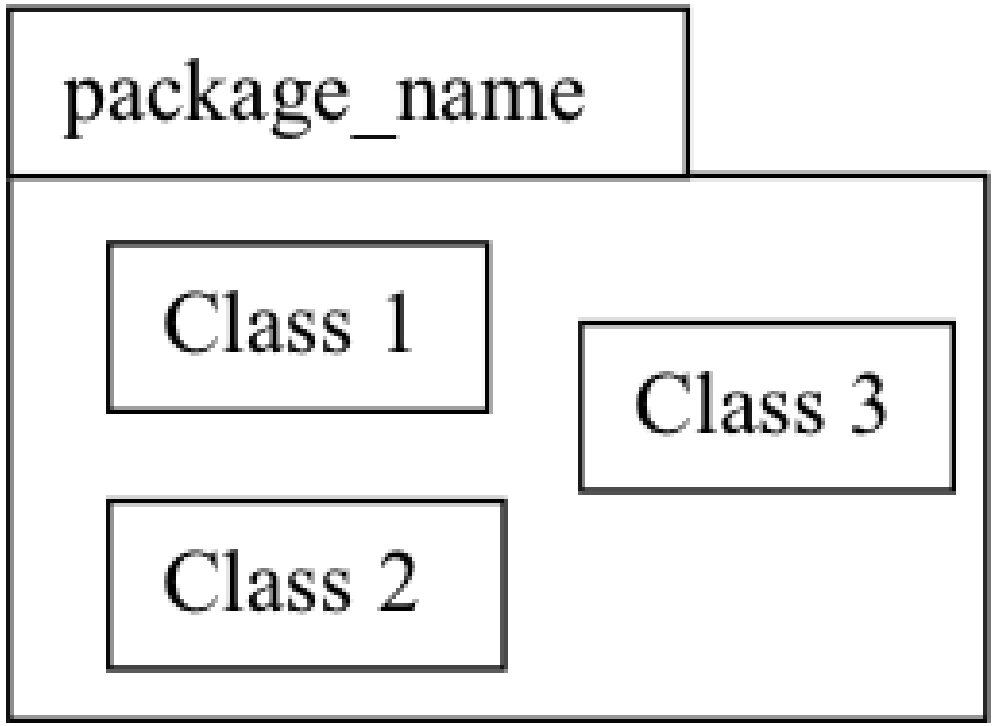
UML Aggregation

- Defines a PART_OF relation
- Differs from IS_COMPONENT_OF
- TRIANGLE has its own methods
- TRIANGLE implicitly uses POINT to define its data attributes



UML Packages

IS_COMPONENT_OF is represented via the package notation



Point ADT Module

Template Module

PointT

Uses

N/A

Syntax

Exported Types

PointT = ?

Point ADT Module Continued

Exported Access Programs

Routine name	In	Out	Exceptions
new PointT	real, real	PointT	
xcoord		real	
ycoord		real	
dist	PointT	real	

Semantics

State Variables

xc: real
yc: real

Point Mass ADT Module

Template Module

PointMassT **inherits** PointT

Uses

PointT

Syntax

Exported Types

PointMassT = ?

Point Mass ADT Module Continued

Exported Access Programs

Routine name	In	Out	Exceptions
new PointMassT	real, real, real	PointMassT	NegMassExceptio
mval		real	
force	PointMassT	real	
fx	PointMassT	real	

Semantics

State Variables

ms: real

Point Mass ADT Module Semantics

new PointMassT(x, y, m):

- transition: $xc, yc, ms := x, y, m$
- output: $out := self$
- exception: $exc := (m < 0 \Rightarrow \text{NegativeMassException})$

force(p):

- output:

$$out := \text{UNIVERAL\_G}\frac{self.ms \times p.ms}{self.\text{dist}(p)^2}$$

- exception: none