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

Winter 2018

12 Object Oriented Design (Ghezzi Ch. 4)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 31, 2018

12 Object Oriented Design (Ghezzi Ch. 4)

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

Administrative Details

- Assignment 1
  - Part 2: January 31, 2018
- Questions?

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 (why?)

Set Idiom (H&S)

Routine name	**In**	**Out**	**Exceptions**
set_add	T		Member, Full
set_del	T		NotMember
set_member	T	boolean	
set_size		integer	

Sequence Idiom (H&S)

Routine name	**In**	**Out**	**Exceptions**
seq_init			
seq_add	integer, T		PosOutOfRange, Full
seq_del	integer		PosOutOfRange
seq_setval	integer, T		PosOutOfRange
seq_getval	integer	T	PosOutOfRange
seq_size		integer	
seq_start			
seq_next		T	AtEnd
seq_end		boolean	
seq_append	T		Full

When would you use seq_next in the interface, and exclude seq_getval?

Tuple Idiom Version 1 (H&S)

Routine name	**In**	**Out**	**Exceptions**
tp_init			
tp_set_f_1	T_1		
tp_get_f_1		T_1	
...	...	...	...
tp_set_f_N	T_N		
tp_get_f_N		T_N	

What is a potential problem with this idiom, especially if there are many fields to the tuple?

Tuple Idiom Version 2 (H&S)

Routine name	**In**	**Out**	**Exceptions**
tp_init			
tp_set	T_1, T_2, ..., T_N		
tp_get		T	

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

In Python, what class do all classes inherit?

What method inherited from object did we recently override?

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

In Python, what class do all classes inherit?

What method inherited from object did we recently override?

Template Module Employee

Routine name	**In**	**Out**	**Except**
New 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

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

emp1 = Administrative_Staff()
emp2 = Technical_Staff()
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff()
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

emp1 = Administrative_Staff() $\surd$
emp2 = Technical_Staff() $\surd$
emp3 = emp1
emp3 = (Technical_Staff) emp1

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

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

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

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

Inheritance Continued

emp1, emp2: Employee
emp3: Technical_Staff

What assignments are allowed? That is, where would polymorphism allow us to switch references to the RHS with what appears on the LHS?

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

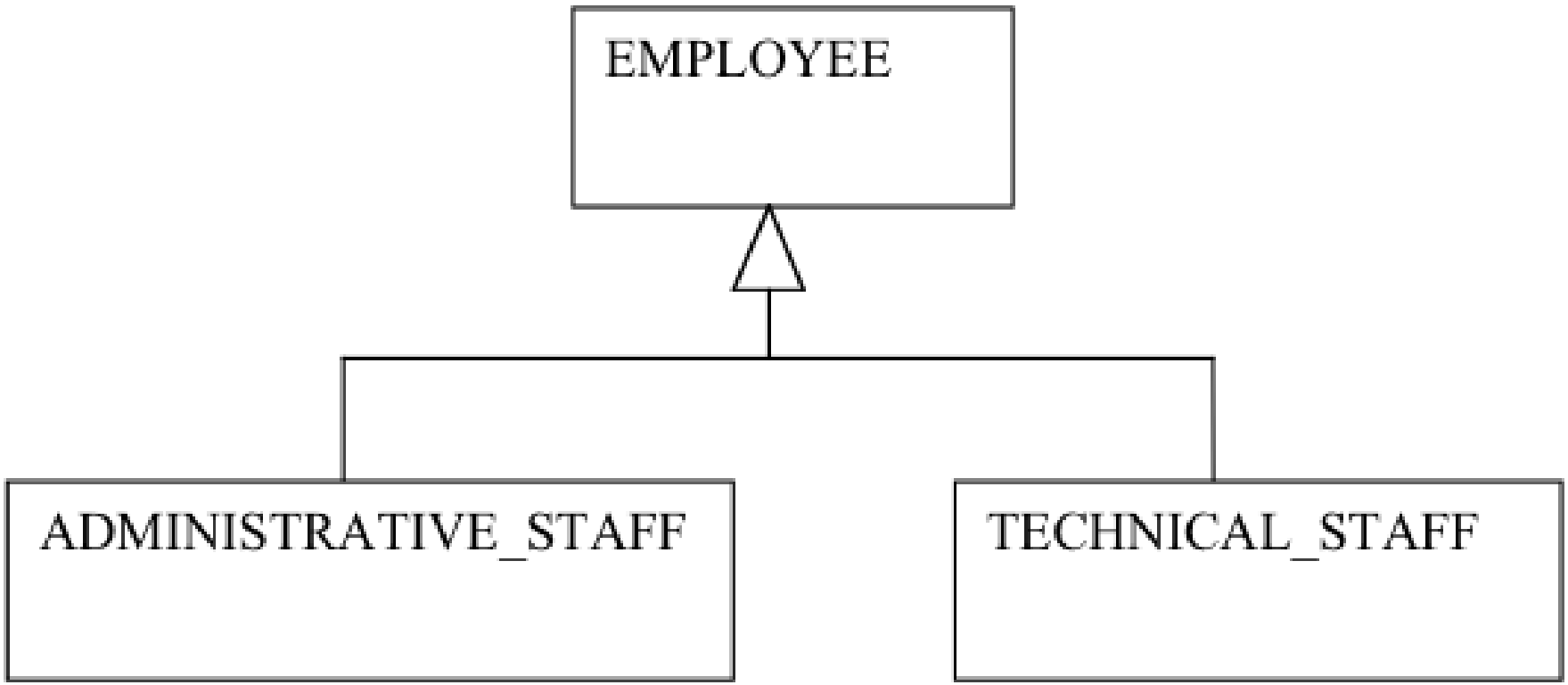
- Many languages, like C, use static type checking
- OO languages use dynamic type checking as the default
- There is a difference between a type and a class once we know this
  - Types are known at compile time
  - The class of an object may be known only at run time

How can Inheritance be Represented?

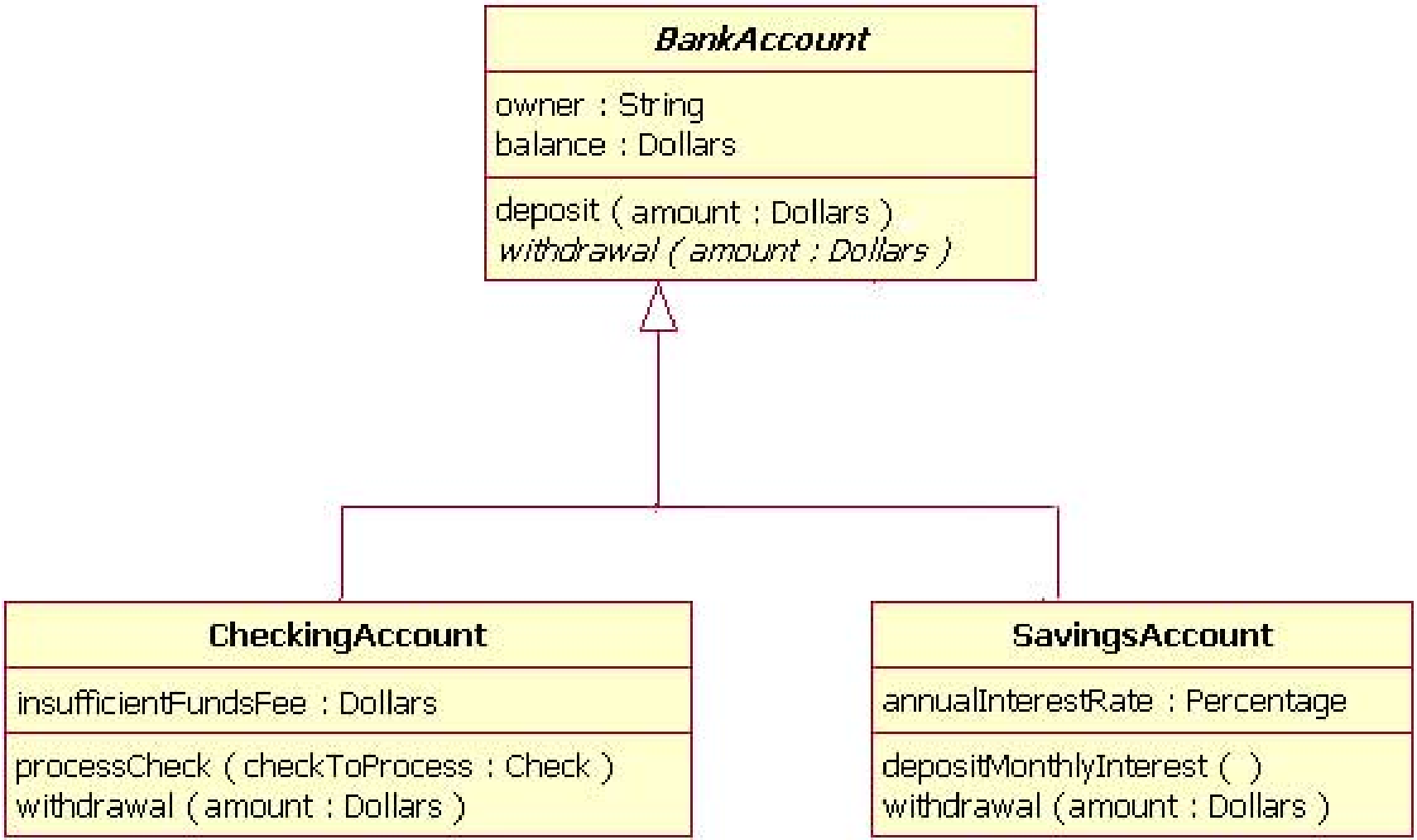
- We start introducing the UML notation
- UML (Unified Modelling Language) is a widely adopted standard notation for representing OO designs
- We introduce the UML class diagram
- Classes are described by boxes

Any guesses on what Parnas said UML stood for?

UML Representation of Inheritance



Bank Account Example



Class Diagram Versus MIS

- What information do the MIS and Class Diagram have in common?
- What information does the MIS add?
- What information does the Class Diagram add?

Class diagrams are closer to code since syntax of methods closer to actual syntax

Class Diagram Versus MIS

- What information do the MIS and Class Diagram have in common?
- What information does the MIS add?
- What information does the Class Diagram add?

Class diagrams are closer to code since syntax of methods closer to actual syntax

Class Diagram Versus MIS

- What information do the MIS and Class Diagram have in common?
- What information does the MIS add?
- What information does the Class Diagram add?

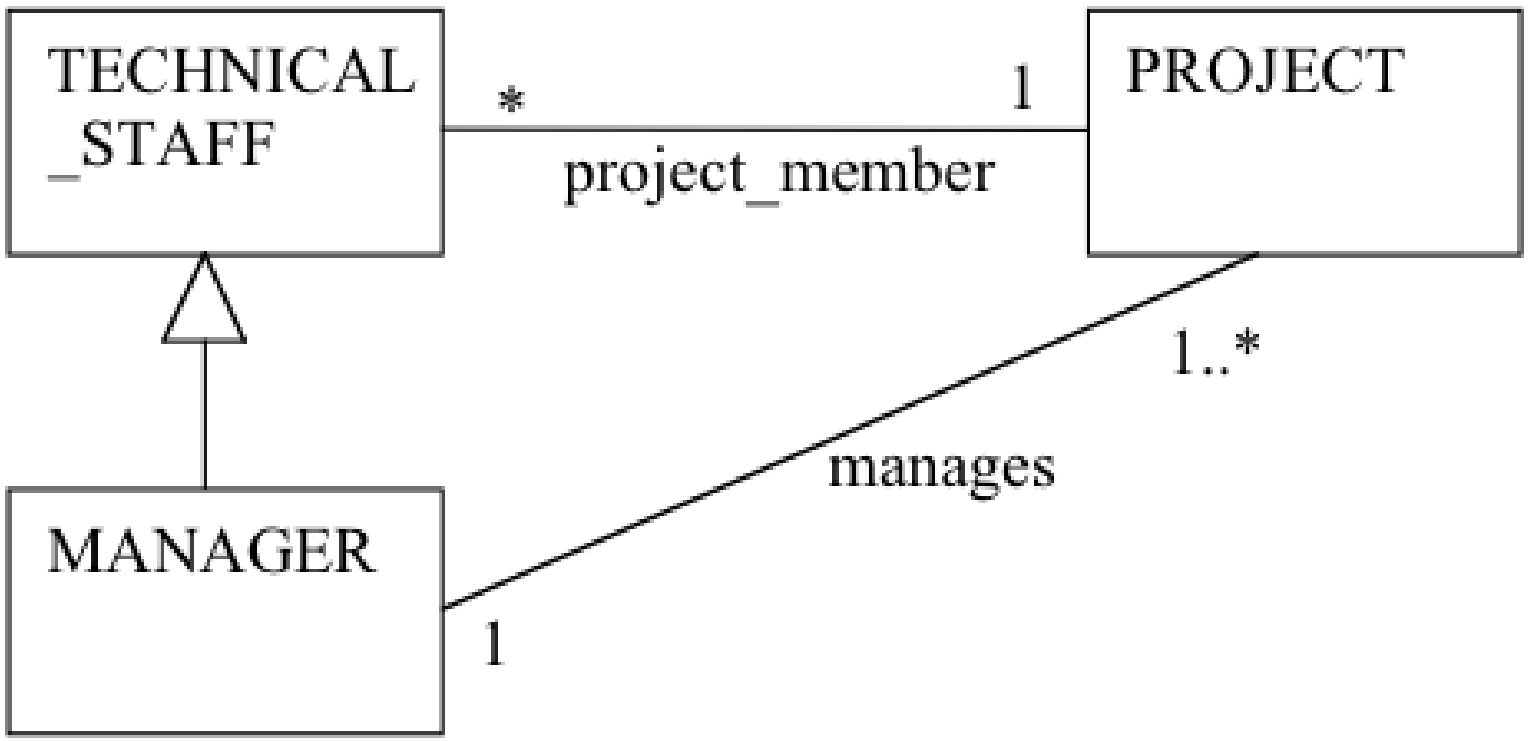
Class diagrams are closer to code since syntax of methods closer to actual syntax

Showing Exceptions in UML Class Diagrams

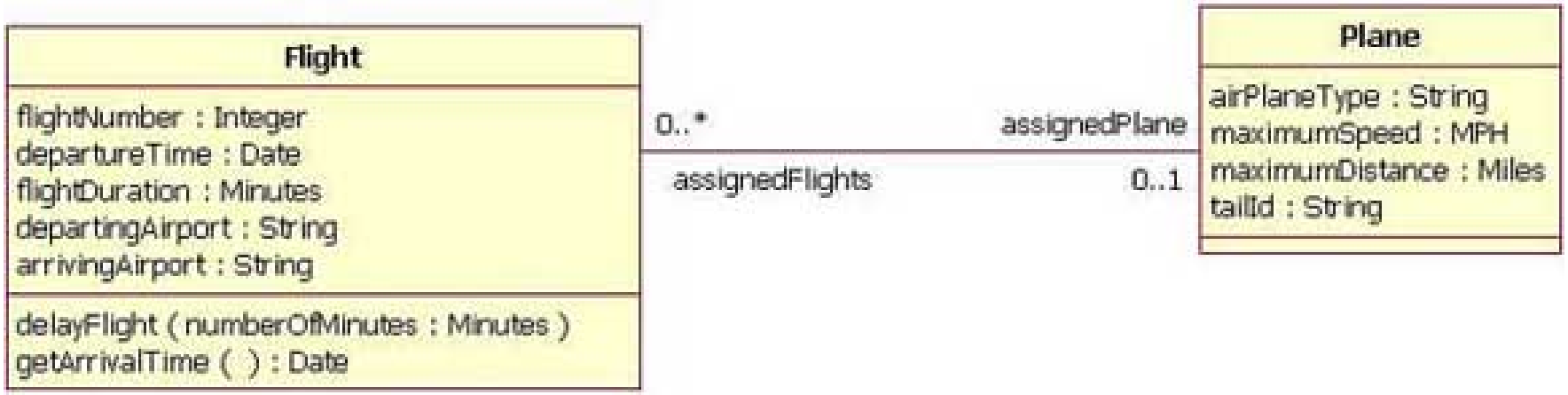
- Usually exceptions are not shown
- If they are, it is in brackets after the method name
- + findAllInstances(): Vector
  {exceptions=NetworkFailure, DatabaseError}

UML Associations

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



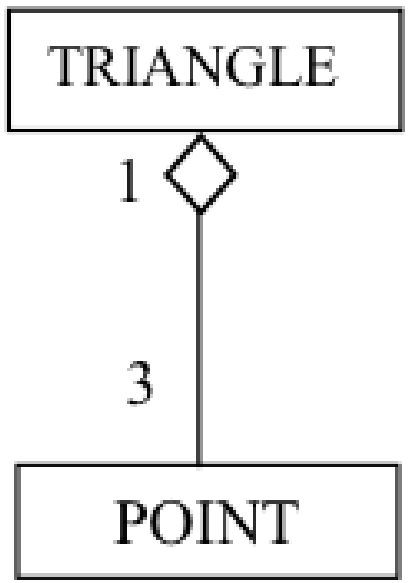
Flight Example



From IBM

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