2009 November 23

Example Test Questions for Inheritance

1.

Give and explain the type rule with genericity using an example method call.

2.

Describe constraint genericity. How is such genericity related to inheritance? Provide an example where constraint genericity is useful.

3.

When you redefine a feature, you can redefine the signature and body independently. Explain why you need redefinition of each part and what are the constraints on redefinition.

4.

When you redefine a feature, there are two parts of a feature you can redefine independently. Explain what they are, why you need redefinition of each part and what are the constraints on redefinition?

5.

What is the rule of change in the context of deciding between use and inheritance? Give three natural language definitions and their corresponding Eiffel class definitions that illustrate the choices in deciding between use and inheritance?

6.

When using multiple inheritance sharing and replication of attributes are the choices that are available. Using a BON diagram, give an example where sharing is not an option. Explain why sharing is not an option.

7.

When using multiple inheritance sharing and replication of attributes are the choices that are available. Using a BON diagram, give an example where sharing and replication are options. Explain why sharing is an option and how you get sharing. Explain why replication is an option and how you get replication.

8.

When using multiple inheritance some features may be joined. Using a BON diagram, give an example where join is an option and explain under what conditions join is possible.

9.

Explain what happens to the export status of inherited features in an heir.

10.

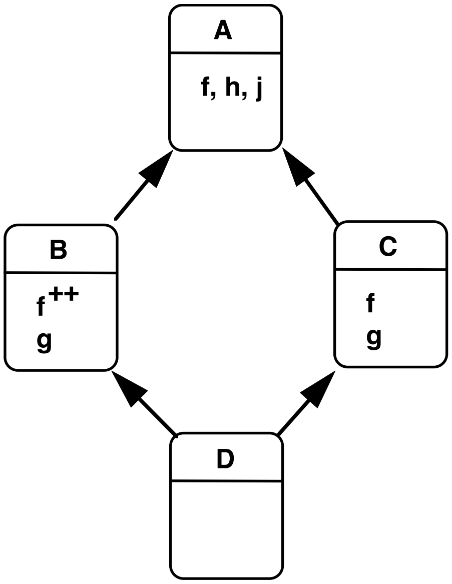
Why would you not normally build the QUEUE class from the ARRAY class using inheritance?

11.

In Eiffel, inherited functions can be redefined as attributes but not vice versa. Why not?

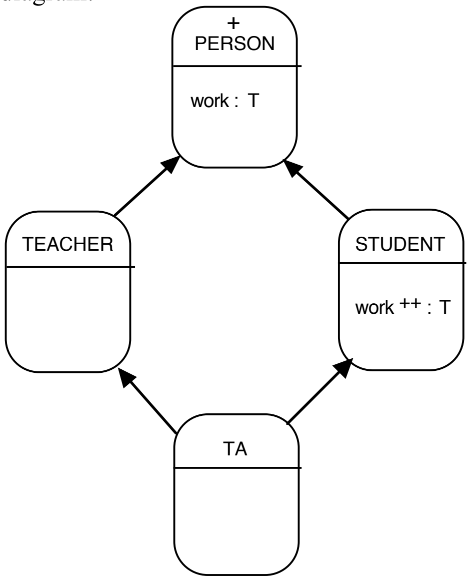
12.

Consider the following inheritance diagram. What problems can arise and how can they resolved? You are not required to solve all problems simultaneously, just describe the various problems and their resolution.



13.

Consider the following BON diagram.



A Give and explain one variation of the Eiffel inheritance and adaptation clauses for the class TA that permit sharing the common feature.

B Give and explain a different variation of the Eiffel inheritance and adaptation clauses for the class TA that permit sharing the common feature.

C Give and explain the Eiffel inheritance and adaptation clauses for the class TA that permit replicating the common feature

D Using an example, explain static and dynamic types of an entity and how they relate.

14.

Explain polymorphism and dynamic binding.

15.

Explain why the creation status of a feature in the parent has no bearing on the creation status of that feature in an heir.

16.

Explain the difference between renaming and redefining an inherited feature.

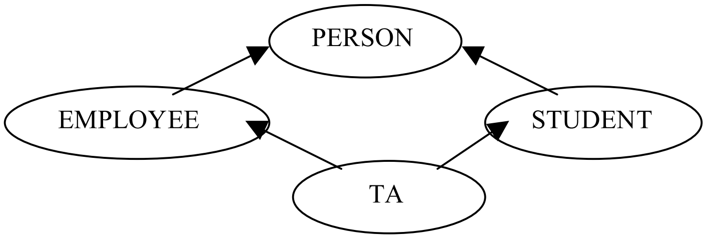
17.

Is the following inheritance scenario correct? If yes, state why. If not, state what is the problem and how it can be resolved?

```
class A[G] feature              class B inherit
      f: G                            A[REAL]
end                                   A[STRING]
                                end
```

18.

Given following class hierarchy:



and declaration and instructions:

```
         p : PERSON; e: EMPLOYEE; s : STUDENT; t1, t2 : TA
         create p.make(…);   create s.make(…);  create t1.make(…)
```

are the following instructions valid? Give a reason for your answer.

```
create PERSON t2.make(…)
create TA e.make(…)
p := t1
s ?= p
```

19.

In the following design, the class STUDENT_TRAINEE is missing feature adaptation or other Eiffel inheritance statements, thereby introducing inconsistencies and errors. There is additional missing program text. Identify what is missing and supply Eiffel program text to correct the program. Assume the given rename and redefine statements are correct.

class PERSON **feature**
eat (what: FOOD) **is**
do print ("I eat sometimes!") **end**
sleep **is do** print ("I sleep sometimes!") **end**
play **is do** print ("I play sometimes!") **end**
end -- class PERSON
	class STUDENT
	inherit PERSON **rename** play **as** study
	redefine eat
	end
	feature
	eat (what: SANDWICH) **is**
	do print ("I eat sandwich!") **end**
	end -- class STUDENT

class EMPLOYEE
inherit PERSON **rename** play **as** work
redefine eat
end
feature
eat (what: STEAK) **is**
	class STUDENT_TRAINEE
	inherit STUDENT **redefine** work, eat
	end
	EMPLOYEE **redefine** eat
	end
	feature

do print ("I eat steak!") **end**
 	
end -- class EMPLOYEE
	eat (what: STEAK_SANDWICH) **is**
	do print ("I eat well!") **end**
	
	end -- class STUDENT_TRAINEE

class FOOD
end -- class FOOD
	class STEAK **inherit** FOOD
	end -- class STEAK

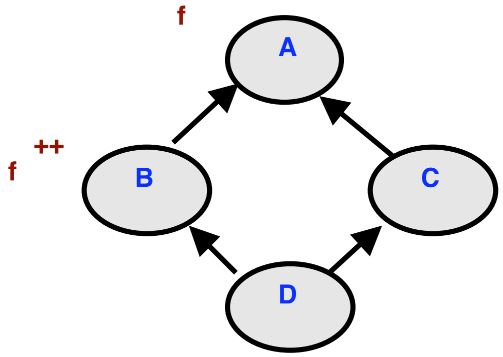
class SANDWICH **inherit** FOOD
end -- class SANDWICH	

20.

The creation inheritance rule states that the creation status of a feature in the parent has no bearing on its creation status in an heir. Explain what this means and why it is a reasonable rule.

21.

In the following diagram class D inherits two versions of feature "f": one from C and a redefined version from B. Based on whether "f " is deferred or not, discuss the problems associated with "f" in D and how they can be resolved.



22.

A Consider the following program text.

class CITY **feature**
 	
the_city: CITY **is**
once
Result := Current
end
 	
invariant
 	
lonely_city: the_city = Current
 	
end -- class CITY
	class LONDON
	
	inherit CITY
	
	end -- class LONDON

	class TORONTO
	
	inherit CITY
	
	end -- class TORONTO

Assuming the class invariant is checked, does the execution of the following statements lead to any assertion violation in Eiffel? Justify your answer.

```
city1, city2 : CITY
toronto: TORONTO
london: LONDON
```

```
create toronto
create london
city1 := london.the_city
city2 := toronto.the_city
```

B Assume class LONDON, from part B, is modified as defined below. Draw memory diagrams for objects created as a result of execution of the set of statements in Part B above. Assume assertions are turned on.

```
class LONDON
inherit CITY
        redefine the_city end
feature
   the_city: LONDON is once Result := Current end

end -- class LONDON
```

23.

Consider the following program text.

```
class THING feature

   the_thing: THING is
      once
         Result := Current
      end

invariant

   only_thing: the_thing = Current

end
```

```
class BETTER

   inherit SOMETHING

end
```

```
class WORSE

   inherit SOMETHING

end
```

Assuming the class invariant is checked, does the execution of the following statements lead to any assertion violation in Eiffel? Justify your answer.

```
thing_1, thing_2 : THING
the_worse: WORSE
the_better: BETTER
create the_worse
create the_better
thing_1 := the_better.the_thing
thing_2 := the_worse.the_thing
```

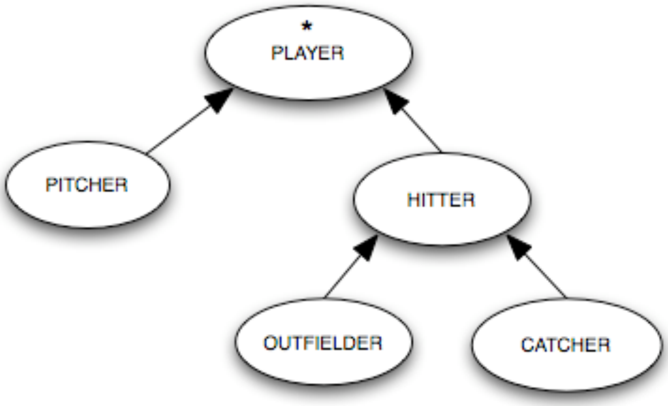
B Assume class BETTER, from part B, is modified as defined below. Draw memory diagrams for objects created as a result of execution of the set of statements in Part B above. Assume assertions are turned on.

```
class BETTER
inherit SOMETHING
        redefine the_thing end
feature
     the_thing: BETTER is once Result := Current end

end
```

24.

The following hierarchy defines players in baseball.

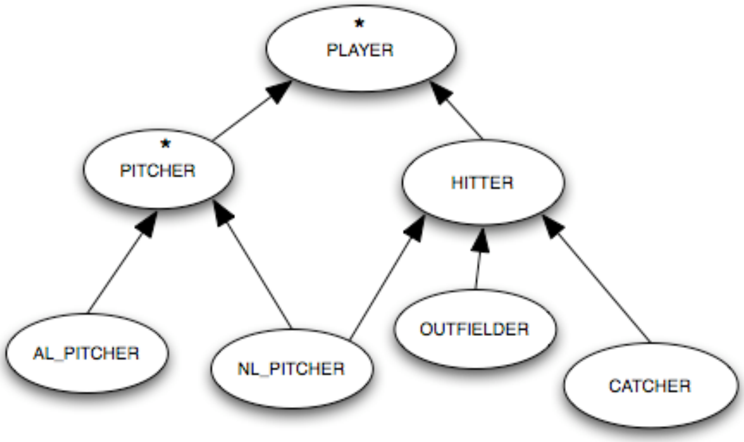


Assume that PITCHER introduces a feature **starts** (indicating the number of games the pitcher started), while the class PLAYER introduces a deferred feature **display** that is implemented by the subtypes.

A A CLOSER is a special kind of pitcher, one who always finishes games. Show the inheritance clause for CLOSER, including and **redefine** and **export** clauses.

B Supose you wanted to modify this design to handle designated hitters, players who only hit and do not play infield or outfield. What changes would you suggest to the hierarchy?

C In the National League, pitchers also hit. Show modifications to the inheritance hierarchy that would take into account National League pitchers and American League pitchers (who do not hit).



D A team has a set of players. Sketch an Eiffel implementation of a routine that would display all the players on a team.

25.

Suppose you wrote the following four classes but you haven’t tried to compile them. Consider how the class **DELTA** can handle possible name clashes.

```
deferred class ALPHA
feature
  one   is do print(“alpha one”) end
  two   is do print(“alpha two”) end
  three is do print(“alpha three”) end
  four  is deferred end
end
```

```
deferred class BETA
  inherit
    ALPHA
      redefine two end
feature
  two is do print(“beta two”)  end
end
```

```
class CHARLIE
  inherit
    ALPHA redefine two, three end
feature
  two   is do print(“charlie two”)
           end
  three is do print(“charlie three”)
           end
  four  is do print(“charlie four”)
           end
end
```

```
class DELTA
  inherit
    BETA
      rename one as  beta_one end
    CHARLIE
      rename one as charlie_one end
      select four
      end
feature
end
```

The answers to the following questions require either a *yes* or *no*, as appropriate, and must give a rationale that references appropriate rules or principles. A *yes* or *no* without a rationale receives a zero grade.

A Is it necessary for class **DELTA** to include the procedure **one** in a **rename** clause? If so, please explain why. If not, please explain why not.

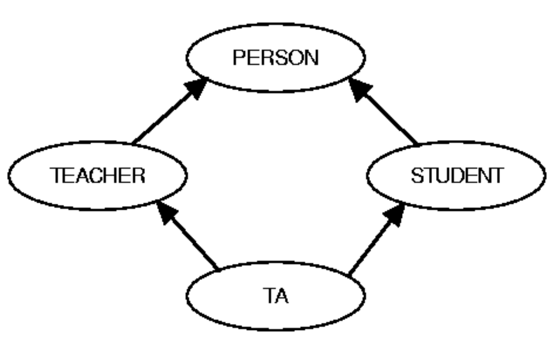
B Is it acceptable for class **DELTA** to not refer to the procedure **two** in a **rename, redefine, undefine,** or **select** clause? If so, you must explain why. If not, you must explain why not, and describe one way in which the problem can be resolved.

C Is it acceptable for class **DELTA** to not refer to the procedure **three** in a **rename, redefine, undefine,** or **select** clause? If so, you must explain why. If not, you must explain why not, and describe one way in which the problem can be resolved.

D Is it necessary for class **DELTA** to refer to the procedure **four** in a **select** clause? If so, please explain why. If not, please explain why not.

26.

Consider the following class hierarchy and program text.



p : PERSON;
t : TEACHER;
s : STUDENT;
ta1, *ta2* : TA
create p.make(…)
create s.make(…)
create ta1.make(…)

For each of the following instructions state whether or not the instruction is valid and give the reason why.

1) **create** {PERSON} *ta2*.make(…)

2) **create** {TA} *t*.make(…)

3) *p* := *ta1*

4) *s* ?= *p*

27.

A) We learned that software modeling is difficult: by changing our view point we can rephrase an "**is-a**" relationship to a "**has-a**" relationship. For example "every software engineer **is an** engineer" can be rephrased as "every software engineer **has an** engineer component", therefore changing "inheritance" relationship to "client-supplier". Discuss what criteria can be used, in such situations, to decide whether is-a or has-a is the most appropriate relation for the chosen entities.

B) We have two relations between classes: client-supplier and inheritance. Compare these two relations in terms of reuse, information hiding and protection against change. That is to say which one of these relationships allow for reuse of services provided by a class, which allows for information hiding, and which protects a class against change. Justify your answers.

C) In Eiffel a descendent can redefine a function with no argument into an attribute, but not vice versa. First explain why it is not possible to redefine an attribute into a function. Then explain how a descendent defining a function into an attribute should handle the postcondition of that function.

28.

A With multiple inheritance sharing and replication of features are the choices that are available. Using a BON diagram, give an example where sharing and replication are options. Explain why sharing is an option and give the Eiffel program text that enables sharing. Explain why replication is an option and give the Eiffel program text that enables replication.

29.

A. With multiple inheritance sharing and replication of attributes are the choices that are available. Using a BON diagram, give an example where sharing is not an option. Explain why sharing is not an option.

30.

Answer the following questions with respect to the program text given at the end of the question.

A What would be the class invariant for class A? Will the class invariant be satisfied immediately after creating an instance of A – create a.make. Justify your answer completely and in detail.

B Assume that all assertions are changed to **true**, what would be the result of executing the program text in the ROOT_CLASS. Justify your answer completely and in detail.

C Is the redefinition of the feature printa in class B correct. Justify your answer in detail by using the complete pre and post conditions to justify your conclusion.

class D **creation** make **feature**	**class** C	**class** ROOT_CLASS
x : INTEGER	**inherit D**	**creation** make
y : INTEGER	**rename** printa **as** c_print	**feature**
	redefine double **end**	
make **is do** x := 2 ; y := 5 **end**		a : A ; b : B
	creation make **feature**	c : C ; d : D
printa(a : INTEGER) **is**		
require y > x	printa(a : INTEGER) **is**	make **is do**
do	**require true**	!! a.make ; !!b.make

io.put_string("In D a is: ")
io.put_integer(a)
ensure y - x > 0
end
 	
double : INTEGER **is**
require y > 2
do
result := 2 * y + x
ensure result > y + x
end
 	
invariant x > 1
end
	do
	io.put_string("In C a is: ")
	io.put_integer(a)
	ensure true
	end
	
	double : INTEGER **is**
	require else y > 0
	do
	result := x + y - x*y
	ensure then result > x + y
	end
	
	invariant y >= 5
	end
		!! c.make ; !!d.make
		
		a.printa(a.double)
		d := a
		d.printa(a.double)
		
		end

class B **inherit D** **redefine** printa, make **end** **creation** make **feature** z : INTEGER make **is** do precursor ; z := -4 **end** printa(a : INTEGER) **is** **require else** x > 0 **do** io.put_string("In B a is: ") io.put_integer(a) **ensure then true** **end** **invariant** z < x **or** z > y y – 5 > x **end**	**class** A **inherit** B **rename** printa **as** b_print **redefine** double, make **end** C **undefine make** **redefine** double **select** c_print **end** **creation** make **feature** make **is do** precursor double := z + x + y **end** double : INTEGER **invariant** ??? **end**	

31.

Eiffel has four mechanisms for adaptation. Describe and give an example of each one.

32.

A. What is the rule of change in the context of deciding between use and inheritance?

B. Give three natural language definitions and their corresponding Eiffel class definitions that illustrate the choices in deciding between use and inheritance?