UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

MSci Honours Degree in Mathematics and Computer Science Part IV
MEng Honours Degrees in Computing Part IV
MSc in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the
Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the
Associateship of the Royal College of Science

PAPER C471

ADVANCED ISSUES IN OBJECT ORIENTED PROGRAMMING

Tuesday 23 April 2002, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

1 The language L2 is outlined on pages 5-7 of this paper.

a Consider the following program P (assume that L2 was extended with integers with the usual meaning, and with sequential expressions):

```
class B{}
class A{ A next; int i; m(A x){ this.next = x} }
```

What is the contents of the store after evaluation of the following expressions in a store where **this** is mapped to an object of class B?

```
x = new A(); x.i=1; x.m(x);
```

b Consider an extension of L2 which supports object cloning. The method **clone()** is applicable to all objects, and creates a shallow copy of the receiver. For example:

```
x = new A(); x.i = 1; x.m(x);
x = x.clone();
x.i = 2;
x.i;                // returns 2
x.next.i;           // returns 1
```

i) Write out the store after execution of the above.
ii) Extend the operational semantics of L2 to describe the above.
iii) Extend the type system of L2 to describe the above.

c Consider the following classes with overloaded methods:

```
class A{}
class B extends A{ }
class C{
     int m(A x){1}
     int m(B x){2}     }
class D extends C{
     int m(A x){3}
     int m(B x){4}   }
```

i) Assuming overloading as in Java or C++, what results do the following expressions produce?

```
A a1; B b1; C c1; D d1;
a1 = new B; b1 = new B; c1 = new D; d1 = new D;
c1.m(a1);
c1.m(b1);
d1.m(a1);
d1.m(b1);
```

ii) Modify L2, so that method selection is according to the class of the receiver *and* the argument, and so, `c1.m(a1)` in the above example returns 4.

iii) According to the new semantics, what do the other three expressions return?

iv) Why do Java and C++ not implement the approach described in ii?

The three parts carry, respectively, 20%, 30% and 50% of the marks.

2a i) What is multiple inheritance? (in one sentence)

ii) What is the difference between virtual, and non-virtual multiple inheritance? (in one sentence)

iii) Give a practical example where virtual multiple inheritance is preferable (in one sentence). Give a practical example where non-virtual multiple inheritance is preferable (in one sentence).

b Consider the following C++ classes:

```
class A{ public:
   int i;
   virtual int f( ){ return 2*i; } };

class B1: public A{ public:
   virtual int g( ){ return i + f(); }
   void set(int x){ i=x;   }  };

class C1: public A{ public:
   virtual int f( ){ return 5*i; }
   void set(int x){ i=x;   }  };

class D1: public B1, public C1 {    };
```

i) What does `d1->f()` return after execution of the following statements?

```
B1* b1; C1* c1; D1* d1;
d1=new D1(); b1=d1; c1=d1;
c1->set(10); b1->set(20);
```

ii) Give the representation of a `D1` object.

iii) Give the representation of the body of `B1::g()`.

iv) Give the representation of the assignments `b1=d1; c1=d1`.

c Consider the following C++ classes:

```
class B2: public virtual A{ public:
   virtual int g( ){ return i + f(); }
   void set(int x){ i=x;   }  };

class C2: public virtual A{ public:
   virtual int f( ){ return 5*i; }
   void set(int x){ i=x;   }  };

class D2: public B2, public C2 {    };
```

i) What does `d2->f()` return after execution of the following statements?

```
B2* b2; C2* c2; D2* d2;
d2=new D2(); b2=d2; c2=d2;
c2->set(10); b2->set(20);
```

ii) Give the representation of a `D2` object.

iii) Give the representation of the body of `B2::g()`.

iv) Give the representation of the assignments `b2=d2; c2=d2`.

The three parts carry, respectively, 20%, 40% and 40% of the marks.

3 This question is about verification and dynamic linking in Java.

a Which three properties does successful verification guarantee?

b i) For which Java constructs does the Java verifier check subtypes?

ii) For which Java constructs does the Java verifier not check subtypes?

iii) For which reasons does the verifier not check subtypes for the constructs you mentioned in ii)?

c Consider the following piece of code

```
interface I   {   }
interface J extends I   {   }
class K implements J   {   }

class A{ }
class B extends A { }
class C extends B { }

class D extends B {
        void m1(C c){ m2(c); }
        void m2(B b){   }          }

class E {
        A m1(K k){ return m2(k);   }
        B m2(I i){ return new B(); }         }

class F extends E {
        void m1()    {
            System.out.println( " --3 " );
        B b = new D(); }          }

class Test{
        public static void main(String[] args){
                System.out.println( " --1 ");
                F f = new F();
                System.out.println( " --2 ");
                f.m1();
                System.out.println( " --4 ");   }
```

i) What is the output of the method `main` with the verifier on, in verbose mode (i.e. with notification of loading of classes)?

ii) What is the output of the method `main` with the verifier off, in verbose mode (i.e. with notification of loading of classes)?

The three parts carry, respectively, 15%, 35% and 50% of the marks.

4a List the main features of a class based language, like SELF.

b Assume that SELF has predefined boolean values *true* and *false*, with a negation operation *not*, and that it has predefined arithmetic values, with increment operation ++.
Consider an object *o1*, which has a boolean state which is *false*, and a method *ping*, which negates its state.
Consider objects *o2*, *o3*, which have a boolean state which is *false*, a method *ping*, which negates the particular object's state and increments a counter; the counter counts how many times any of the two objects *o2*, *o3* received the method *ping*.

i) Draw a diagram representing *o1* in SELF without encoding of classes.

ii) Draw a diagram representing *o1* in SELF with encoding of classes.

iii) Draw a diagram representing *o2*, *o3* in SELF.

c Consider the $\varsigma\lambda$ –calculus, with terms defined by

a, b	$::= x$	*a variable*
	$\mid\ [\ l_i = \varsigma(z_i)\, b_i\ ^{i=1..n}\ ]$	an object
	$\mid\ b.l \Leftarrow \varsigma\,(z)\, a$	method override
	$\mid\ b.l$	method call
	$\mid\ a\ b$	function application
	$\mid\ \lambda x(a)$	function

and evaluation, for terms a and o, where $o \equiv [\ l_i = \varsigma(z_i)\, b_i\ ^{i=1..n}\ ]$ (l_i distinct) defined by

$$\lambda x(a)b \rightarrow a\,\{\{ x \leftarrow b \}\}$$

$$o.\,l_j \rightarrow b_j\,\{\{ x_j \leftarrow o \}\} \qquad (j \in 1..n)$$

$$o.\,l_j \Leftarrow \varsigma(y)b \rightarrow [\ l_j = \varsigma\,(y)b,\ l_i = \varsigma\,(z_i)b_i\ ^{i=(1..n)\setminus j}\ ] \qquad (j \in 1..n)$$

i) Write out the steps involved in the evaluation of the term *counter0.tick.cont*, where *counter0* is defined as

$$counter0 \equiv [\ cont = \varsigma(x)0,\ \ tick = \varsigma(y)y.cont \Leftarrow \varsigma(z)y.cont + 1\ ]$$

ii) Write out the steps involved in the evaluation of the term *cntr0.tick.cont*, where *cntr0* is defined as:

$$cntr0 \equiv [\ cont = \varsigma(x)0,\ \ tick = \varsigma\,(y)\,Counter.tick(y)\ ]$$
$$Counter \equiv [\ tick = \varsigma\,(y)\,\lambda\,(x)\,x.cont \Leftarrow x.cont + 1\ ]$$

iii) Compare the terms *counter0* and *cntr0*. Hint: What role does the term *Counter* play?

The three parts carry, respectively, 10%, 40% and 50% of the marks.

The Syntax of $\mathcal{L}_2$

progr	::=	*class**
class	::=	class *c* extds *c* { *field** *meth** }
field	::=	*type f*
meth	::=	*type m* (*type* x) { *e* }
type	::=	bool | *c*
e	::=	if *e* then *e* else *e* | *var* := *e* | *e* .*m* (*e*)
	|	new *c* | *var* | this | true | false | null .
var	::=	x | *e* .*f*

Field and method lookup functions

For program P, identifiers c, f, m and $\mathcal{C}(\mathsf{P}, \mathsf{c}) = \mathsf{class\ c\ extds\ c'}$ we define:

$$
\begin{array}{lcl}
\mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) & = & \begin{cases} \mathsf{t} & \mathit{if}\ \mathsf{cBody} = \ldots\ \mathsf{t\ f}\ \ldots \\ \mathit{Udf} & \mathit{otherwise} \end{cases} \\
\mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) & = & \begin{cases} \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) & \mathit{if}\ \ \mathcal{FD}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathit{Udf}, \\ \mathcal{F}(\mathsf{P},\mathsf{c}',\mathsf{f}) & \mathit{otherwise} \end{cases} \\
\mathcal{F}(\mathsf{P},\mathsf{Object},\mathsf{f}) & = & \mathit{Udf} \\
\mathcal{F}s(\mathsf{P},\mathsf{c}) & = & \{\mathsf{f} \mid \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}) \neq \mathit{Udf}\} \\
\mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) & = & \begin{cases} \mathsf{t\ m(\ t_1\ x)}\ \phi\{\ \mathsf{e}\ \} & \mathit{if}\ \mathsf{cBody} = \ldots\mathsf{t\ m\ (\ t_1\ x)\ \{\ \ e\ \}}\ldots \\ \mathit{Udf} & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) & = & \begin{cases} \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) & \mathit{if}\ \ \mathcal{MD}(\mathsf{P},\mathsf{c},\mathsf{m}) \neq \mathit{Udf}, \\ \mathcal{M}(\mathsf{P},\mathsf{c}',\mathsf{m}) & \mathit{otherwise} \end{cases} \\
\mathcal{M}(\mathsf{P},\mathsf{Object},\mathsf{m}) & = & \mathit{Udf}
\end{array}
$$

The Operational Semantics of $\mathcal{L}_2$

val

$$\frac{}{\mathsf{v},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma}$$

cond$_1$

$$\frac{\begin{array}{l}\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{true},\sigma''\\ \mathsf{e}_1,\sigma'' \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'\end{array}}{\mathsf{if\ e\ then\ e_1\ else\ e_2},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'}$$

cond$_2$

$$\frac{\begin{array}{l}\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{false},\sigma''\\ \mathsf{e}_2,\sigma'' \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'\end{array}}{\mathsf{if\ e\ then\ e_1\ else\ e_2},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'}$$

var

$$\frac{}{\begin{array}{l}\mathsf{x},\sigma \rightsquigarrow_{\mathsf{P}} \sigma(\mathsf{x}),\sigma\\ \mathsf{this},\sigma \rightsquigarrow_{\mathsf{P}} \sigma(\mathsf{this}),\sigma\end{array}}$$

fld

$$\frac{\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \iota,\sigma'}{\mathsf{e.f},\sigma \rightsquigarrow_{\mathsf{P}} \sigma'(\iota)(\mathsf{f}),\sigma'}$$

ass

$$\frac{\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'}{\mathsf{x{:=}e},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'[\mathsf{x}\mapsto\mathsf{v}]}$$

fldAss

$$\frac{\begin{array}{l}\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \iota,\sigma''\\ \mathsf{e}',\sigma'' \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'''\\ \sigma'''(\iota)(\mathsf{f}) \neq \mathcal{U}df\\ \sigma' = \sigma'''[\iota\mapsto\sigma'''(\iota)[\mathsf{f}\mapsto\mathsf{v}]]\end{array}}{\mathsf{e.f{:=}e}',\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma'}$$

new

$$\frac{\begin{array}{l}\mathcal{F}s(\mathsf{P},\mathsf{c}) = \{\mathsf{f}_1,\ldots,\mathsf{f}_r\}\\ \forall \mathsf{l}\in 1,\ldots,\mathsf{r}: \quad \mathsf{v}_l \text{ initial for } \mathcal{F}(\mathsf{P},\mathsf{c},\mathsf{f}_\mathsf{l})\\ \iota \text{ is new in } \sigma\end{array}}{\mathsf{new\ c},\sigma \rightsquigarrow_{\mathsf{P}} \iota,\sigma[\iota\mapsto[[\,\mathsf{f}_1:\mathsf{v}_1,\ldots,\mathsf{f}_r:\mathsf{v}_r\,]]^{\mathsf{c}}]}$$

null

$$\frac{\mathsf{e},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{null},\sigma'}{\begin{array}{l}\mathsf{e.f{:=}e}',\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{nullPntrExc},\sigma'\\ \mathsf{e.f},\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{nullPntrExc},\sigma'\\ \mathsf{e.m}(\mathsf{e}_1),\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{nullPntrExc},\sigma'\end{array}}$$

methCall

$$\frac{\begin{array}{l}\mathsf{e}_0,\sigma \rightsquigarrow_{\mathsf{P}} \iota,\sigma_0\\ \mathsf{e}_1,\sigma_0 \rightsquigarrow_{\mathsf{P}} \mathsf{v}_1,\sigma_1\\ \sigma_1(\iota) = [[\ \ldots\ ]]^{\mathsf{c}}\\ \mathcal{M}(\mathsf{P},\mathsf{c},\mathsf{m}) = \mathsf{t\ m(\ t_1\ x)\ \ \{\ e\ \}}\\ \sigma' = \sigma_1[\mathsf{this}\mapsto\iota][\mathsf{x}\mapsto\mathsf{v}_1]\\ \mathsf{e},\sigma' \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma''\end{array}}{\mathsf{e}_0.\mathsf{m}(\mathsf{e}_1),\sigma \rightsquigarrow_{\mathsf{P}} \mathsf{v},\sigma''[\mathsf{this}\mapsto\sigma(\mathsf{this}),\mathsf{x}\mapsto\sigma(\mathsf{x})]}$$

The Type System of $\mathcal{L}_2$

litVarThis

$$\frac{P \vdash \Gamma \diamond}{\begin{array}{l} P,\Gamma \vdash \mathsf{true} : \mathsf{bool} \\ P,\Gamma \vdash \mathsf{false} : \mathsf{bool} \\ P,\Gamma \vdash x : \Gamma(x) \\ P,\Gamma \vdash \mathsf{this} : \Gamma(\mathsf{this}) \end{array}}$$

newNull

$$\frac{\begin{array}{l} P \vdash \Gamma \diamond \\ P \vdash c \diamond_c \end{array}}{\begin{array}{l} P,\Gamma \vdash \mathsf{null} : c \\ P,\Gamma \vdash \mathsf{new}\ c : c \end{array}}$$

fld

$$\frac{\begin{array}{l} P,\Gamma \vdash e : c \\ \mathcal{F}(P,c,f) = t \end{array}}{P,\Gamma \vdash e.f : t}$$

ass

$$\frac{\begin{array}{l} P,\Gamma \vdash x : t \\ P,\Gamma \vdash e : t' \\ P \vdash t' \leq t \end{array}}{P,\Gamma \vdash x{:=}e : t'}$$

fldAss

$$\frac{\begin{array}{l} P,\Gamma \vdash e : c \\ P,\Gamma \vdash e' : t' \\ \mathcal{F}(P,c,f) = t \\ P \vdash t' \leq t \end{array}}{P,\Gamma \vdash e.f{:=}e' : t'}$$

cond

$$\frac{\begin{array}{l} P,\Gamma \vdash e : \mathsf{bool} \\ P,\Gamma \vdash e_1 : t_1 \\ P,\Gamma \vdash e_2 : t_2 \\ P \vdash t_i \leq t \text{ for } i \in 1,2 \end{array}}{P,\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : t}$$

methCall

$$\frac{\begin{array}{l} P,\Gamma \vdash e_0 : c \\ P,\Gamma \vdash e_1 : t_1' \\ \mathcal{M}(P,c,m) = t\ m(\ t_1\ x)\ \{\ e\ \} \\ P \vdash t_1' \leq t_1 \end{array}}{P,\Gamma \vdash e_0.m(\ e_1) : t}$$