Read completely and carefully the following instructions before starting working on the exam:

1. Write your last and first name on every page that contains parts of your solutions. Use a ballpoint pen or a fountain pen (no pencil). Do not use a red pen. Return the instructions, the tasks, and your solutions.

2. It is neither allowed to use your own papers, documents, scripts, etc., nor any electronic equipment (notebook computers, calculators, cell phones, etc.)

3. Write all your solutions in English.

4. Explain your solutions carefully if a task asks for an explanation.

5. You have 1 hour to complete the exam.

6. Place your student ID on the desk.

Good Luck!

<table>
<thead>
<tr>
<th>Task</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. points</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>40</td>
</tr>
<tr>
<td>Achieved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task 1  Multiple Choice - 9 points

In all of the following questions exactly one answer is correct. To get points, you need to mark only the correct answer. In all other cases you will get no points.

A (3 points)  Subtyping

Assume we add an otherwise clause to method contracts in Java, which gives a condition on the state after the method throws an exception. The implementation of the method has to guarantee that the condition in the otherwise clause is true whenever the method returns exceptionally (that is, via throwing an exception).

Consider a class with an integer field f and the following Java method and its precondition and an otherwise-clause (reminder: final parameters cannot be assigned to):

```java
/// requires n > 0
/// otherwise f < 0
void foo(final int n) throws IOException
```

Assume method foo is overridden in a subclass and that we do not use specification inheritance. Which of the following functions ...

1. ... override foo correctly based on the variance rules of Java and

2. ... have preconditions and otherwise-clauses that would be allowed if the subclass should be a behavioral subtype?

For this, decide what kind of relationship between otherwise-clauses of behavioral subtypes should exist, basing your decision on the substitution principle.

For this exercise, assume FileNotFoundException <: IOException <: Exception and that there is no integer overflow.

(a) requires n == 0
   otherwise f == -1
   void foo(final int n) throws FileNotFoundException

(b) requires n > 0
   otherwise f * f > 0
   void foo(final int n) throws IOException

(c) requires n >= 0
   otherwise f < -n
   void foo(final int n) throws Exception

(d) CORRECT:

   requires n != 0
   otherwise f < -n * n
   void foo(final int n) throws IOException

(e) None of the above would be allowed

Answer:  

   a  b  c  d  e
B (3 points)  Method Binding

Consider the following Java classes:

class A {
    public void foo (Object o) { System.out.println("A"); }
}
class B {
    public void foo (String o) { System.out.println("B"); }
}
class C extends A {
    public void foo (String s) { System.out.println("C"); }
}
class D extends B {
    public void foo (Object o) { System.out.println("D"); }
}
class Main {
    public static void main(String[] args) {
        A a = new C(); a.foo("Java");
        C c = new C(); c.foo("Java");
        B b = new D(); b.foo("Java");
        D d = new D(); d.foo("Java");
    }
}

What is the output of the execution of the method main in class Main?

(a) The code will print A C B D
(b) CORRECT: The code will print A C B B
(c) The code will print C C B B
(d) The code will print C C B D
(e) None of the above

Answer:  a | b | c | d | e
C (3 points)  Bytecode Verification

The method $f$ of class $C$ has the following signature:

```java
def void f(int i, X x);
```

and one local variable $v$. The maximal stack size is equal to 2.

Consider the following Java byte code for the body of method $f$:

```java
0: iload 1
1: istore 3
2: iload 3
3: iconst 5
4: add
5: ifeq 9
6: aload 2
7: astore 3
8: goto 5
9: return
```

Which of the following is correct:

(a) The byte code verifier can successfully verify the code.

(b) The code cannot be verified because it might contain an infinite loop.

(c) The code cannot be verified since instruction 7 stores an object reference in an integer register.

(d) The code cannot be verified due to a stack overflow.

(e) CORRECT:  None of the above.

Answer:  a  b  c  d  e
Task 2  Traits and Behavioral Subtyping - 10 points

Assume the following Scala classes and traits:

```scala
class A {
    def m(a:Int):Int = { return a; }
}

trait B extends A {
    override def m(a:Int):Int = { return super.m(a) * super.m(a); }
}

trait C extends A with B {
    override def m(a:Int):Int = { return super.m(a) / 2 + 1; }
}

trait D extends A with B {
    override def m(a:Int):Int = { return super.m(a) + 2; }
}

class X extends A with D
class Y extends A with D with C
```

A (4 points)
What is the result of calling `m(2)` on an instance of each of `X` and `Y`?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X: 6</td>
<td>Y: 4</td>
</tr>
</tbody>
</table>

B (6 points)

Assume the following specifications:

- A.m
  - ensures true
- B.m
  - ensures result >= a
- C.m
  - ensures result > a*a/2
- D.m
  - ensures result > a*a

Assume our language applies specification inheritance along the linearization order. Does the implementation of `m` in `X` or in `Y` violate its effective contract? For each implementation that violates the contract, write the effective contract and provide an input to `m` for which the method violates that contract.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| Y violates its contract: The effective contract is
result >= a && result > a*a/2 && result > a*a |

The computed value for `m(2)` is 4, but the effective postcondition requires 5 or larger.
Task 3  Multiple Inheritance - 10 points

Consider the following C++ program (recall that default constructors, i.e. constructors without arguments, in C++ do not need to be called explicitly):

class A {
  public:
    A() { a = 1; }
    int a;
};
class B : public A {
  public:
    B() { a += 1; }
};
class C : public A {
  public:
    C() { a *= 2; }
};
class D : public B, public C {};

int main() {
  D* d = new D();
  std::cout << d->a;
  return 0;
}

A (2 points)
The code is rejected by the compiler. Briefly explain why.

solution

The code is rejected by the compiler because the reference d->a is ambiguous.

B (3 points)
Solve the problem by changing only the class signatures, without removing classes from the inheritance declarations.

solution

The problem can be solved using virtual inheritance:

class B : public virtual A {
  public:
    B() { a += 1; }
};
class C : public virtual A {
  public:
    C() { a *= 2; }
};

What is the output of running the program after your changes?
C (5 points)

For this part, you can choose to start with either the original class signatures or your solution from part B. State your choice.

We now add the following public method to A:

```cpp
virtual int get() { return a; }
```

and the following two classes:

```cpp
class X : public A {};
class Y : public D, public X {};
```

and we write a new implementation of the method `main()`:

```cpp
int main() {
    Y* y = new Y();
    B* b = y;
    b->a = rand(); // non-deterministic value assignment
    assert (b->a == y->get());
    return 0;
}
```

The other classes B, C, and D are unmodified.

The code is rejected by the compiler. Briefly explain why.

solution

The code is rejected by the compiler because the reference `y->get()` is ambiguous.

Solve the problem only by adding code to the body of class `Y`. Your solution should make the assertion valid.

solution

One possible solution is to modify the class `Y` as follows:

```cpp
class Y : public D, public X {
public:
    int get() { return B::a; }
};
```

This solution is independent from the choice between original and modified class signatures.
Task 4  Generics - 11 points

Consider the following Java program, which is rejected by the Java compiler:

```java
class Logger<T> {
    public void log(T t) {
        System.out.println(t.loggerString());
    }
}
```

A (2 points)
If the code above were allowed to compile without errors, what could go wrong? To answer, write a brief code sample that uses Logger in a way which causes a failure at runtime.

```java
Logger<Object> l = new Logger<Object>();
l.log(new Object());
```

B (2 points)
How can we fix the class Logger so that it compiles, while preserving its functionality? You should not modify the method log, but otherwise can change or add any code. Your solution should include all details required to check that Logger is a valid Java class.

```java
interface Loggable {
    String loggerString();
}
class Logger<T extends Loggable> {
    ...
}
```

C (2 points)
Assume that class Logger has been fixed to resolve the problem from point A. Let A and B be two classes such that A is a supertype of B and Logger<A> and Logger<B> are valid instantiations. Consider the following method:

```java
void foo(Logger<A> logA) {
    Logger<B> logB = logA;
    logB.log(new B());
}
```

The Java compiler rejects this code. Is the code safe? That is, if it were allowed to compile, would it run without failure?

```java
solution
Yes, the code is safe.
```

D (5 points)
Suppose we relax the Java type system rules to allow contravariant generics.

- Will the method foo compile now?
• What are two situations that will require dynamic checks in order to enable contravariant generics in a language, without limiting what a developer can write in a generic class?

solution

− When calling methods of generic classes, it would be necessary to check whether the dynamic type of the result is a subtype of the static type of the variable where the result is stored.

− When reading fields of generic classes, it would be necessary to check whether the dynamic type of the field is a subtype of the static type of the variable where the object is stored.