Today's Process

- If you haven't completed the solution sheet for Worksheet B, please leave (and go finish it)
- Make sure your student ID (or name) is on your solution sheet
- We'll collect them all, shuffle them, and hand them out – if you get your own, let us know ASAP, since grading your own is not allowed
- Then use a post-it to put your student ID (and name) on the sheet you are grading – otherwise we cannot give you the extra credit you should earn

CSE 331
SOFTWARE DESIGN & IMPLEMENTATION
WORKSHEET B

Autumn 2011

True/False

2

- A UML class diagram can be executed (just like a Java program can be executed).
  - 2 points for **false**
  - 0 points for **true**
- Although some aspects of UML diagrams have great similarity to a programming language, it is not a programming language

True/False

3

- Reducing the size of a test case is an important step in debugging.
  - 2 points for **true**
  - 0 points for **false**
- The objective of this step is to narrow the part of the program that must be considered – and the program is usually much bigger than the size of the test case that failed

True/False

4

- A good practice is to treat all Java exceptions -- both checked and unchecked exceptions -- in the same way.
  - 2 points for **false**
  - 0 points for **true**
- Unchecked exceptions (like NullPointerException) can in principle occur anywhere; checked exceptions (defined for a given program) cannot
  - It's essential to treat checked exceptions carefully, usually through exception chaining – but other approaches can be used for handling unchecked exceptions

True/False

5

- An advantage of implementing `repOK` as a method instead of as an exception is that it allows the implementer of a class to re-establish a broken representation invariant.
  - 2 points for **true**
  - 0 points for **false**
- Doesn't always happen, but it's certainly feasible

True/False

6

- The singleton pattern and the ability to define multiplicity in UML provide the same power to a design/programmer.
  - 2 points for **false**
  - 0 points for **true**
- Both relate to multiplicity, but UML allows much richer relationships than "precisely one instance."

True/False

7

- Representation invariants would be more appropriate to apply to a UML class diagram than in a UML sequence diagram.

  - 2 points for **true**
  - 0 points for **false**

- Representation invariants address the question of "what are legal values of the representation" but do not directly address the question of the operations, their order of invocation, etc.

True/False

8

- Regression testing in principle addresses the removal of tests that no longer apply to a program.

  - 2 points for **true**
  - 0 points for **false**

- Regression testing can no longer run tests that apply to (for example) removed features
  - These tests are often not actually removed, but their failures are noted without concern

True/False

9

- Covariance/contravariance are concepts used to define the type system of a programming language.

  - 2 points for **true**
  - 0 points for **false**

- These terms refer to a relationship among types – they are used to define a languages notion of "stronger" and "weaker" types, allowing or disallowing substitution

True/False

10

- 2 points for **true**
- 0 points for **false**

- At run-time, you cannot determine the precise value of a parameter of a generic class.

Type erasure

- All generic types become type `Object` once compiled
  - One reason: backward compatibility with old byte code
  - So, at runtime, all generic instantiations have the same type

```
List<String>  lst1 = new ArrayList<String>();
List<Integer> lst2 = new ArrayList<Integer>();
lst1.getClass() == lst2.getClass()  // true
```

- You cannot use `instanceof` to discover a type parameter

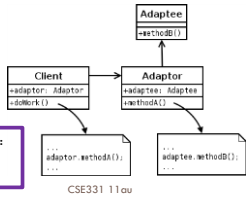
```
Collection<?> cs = new ArrayList<String>();
if (cs instanceof Collection<String>) {
  // illegal
}
```

True/False

11

- Consider the UML class diagram here of the adapter pattern. Does this diagram represent conventional call-return flow-of-control, or does it represent inversion-of-control? In one sentence, justify your answer.

  - -2 points for answering "inversion-of-control"
  - -1, -2, -3 points for missing, confusing or inaccurate justification.



Example: "This pattern uses standard flow-of-control: in each case, the caller explicitly knows the name of the class/interface that it is calling."

12

- Sketch a UML class diagram that describes the relationships among parties, tables, and the waiting list from A3 (Restaurant)
- Key points
  - Waiting list: 0 or more parties (ordered)
  - Table: 0 or more parties
  - Party: Size, name, seated or on waiting list

13

- True or false: The primary objective of design patterns is to make it easier to ensure correctness of an implementation. In one sentence, justify your answer.

- -2 points for **true**
- -1 for a justification not mentioning "change"
- -1, -2 points for additional missing, confusing or inaccurate justification.

Example: "Although a few patterns (such as Singleton) constrain a program in a way that eases reasoning, most patterns (such as Visitor, MVC, etc.) provide ways to ease future program modifications."

Willard Van Orman Quine ⓘ

14

- Famous philosopher and logician (1908-2000)
- Erdős number: 3
  - Same as me: Notkin→Beame→Saks→Erdős
- Two students famous for reasons other than philosophy or logic

Quine: A program that prints itself

15

```
public class Qu {
 public static void main(String[] args) {
 String[] str = {
"public class Qu {",
" public static void main(String[] args) {",
" String[] str = {",
" };",
" for(int i=0;i<3;i++)System.out.println(str[i]);",
" for(int i=0;i<9;i++)System.out.println((char)34+str[i]+(char)34+',');",
" for(int i=3;i<9;i++)System.out.println(str[i]);",
" }",
"}",
 };
 for(int i=0;i<3;i++)System.out.println(str[i]);
 for(int i=0;i<9;i++)System.out.println((char)34+str[i]+(char)34+',');
 for(int i=3;i<9;i++)System.out.println(str[i]);
 }
}
```