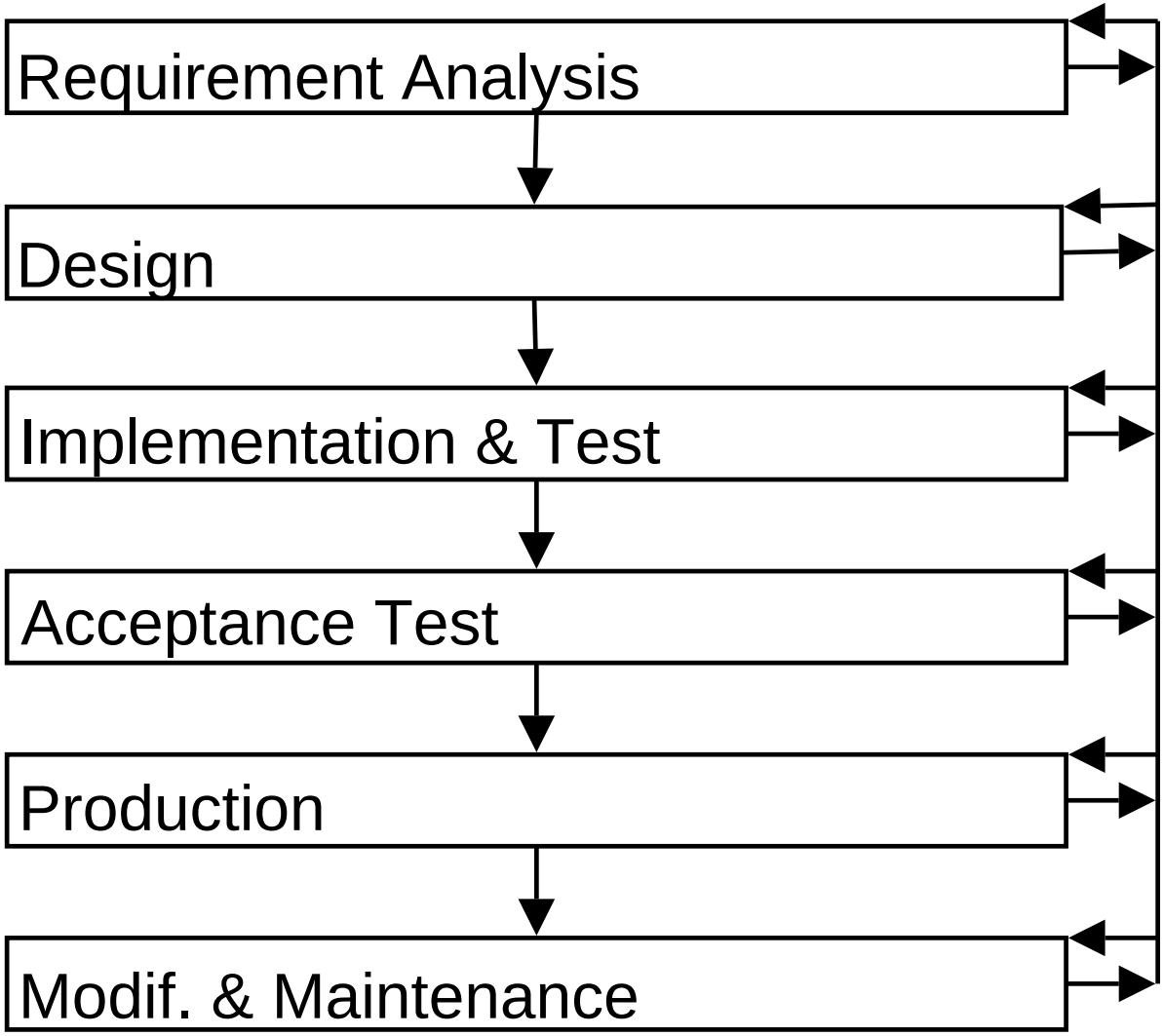
Comp-303 : Programming Techniques
Lecture 15

Alexandre Denault
Computer Science
McGill University
Winter 2004

Last lecture . . .

- Testing is a way of validating correctness of your code.
- Black-box testing is generated from the specification. It always remains, even when implementation changes.
  - check boundary conditions
  - check each path through the specifications
- Glass-box testing complements BB-testing by testing each path in your code.
  - all branches in a conditional
  - 0,1,2 iterations
  - 0 and 1 recursive call
- Debugging allows you to find and correct errors using the scientific method.
  (analyze data, formulate hypothesis, try to disprove)

The traditional software life cycle



Elements of the software life cycle

- Requirement Analysis: The customer desires some services. Those service are analyzed. The result is the requirements document.
- Design : The required program is broken down (decomposed) into modules.
- Implementation & Test: Modules are then implemented.
  - Individual modules are verified with unit tests.
  - Multiple modules are verified with integration tests.

Elements of the software life cycle

- Acceptance Test: We evaluate the program independently of design. In other words, we test using the requirements document. This is sometimes done by a 3rd party. A trial is sometimes run un the real environment.
- Production : The program enters its useful life.
- Modification & Maintenance: Errors can still occur. We fix them as we find them (maintenance). Requirements can also change. These require modification of the existing program.

Waterfall vs. Spiral

- Ideally, each phase is completed before next phase is started. We can thus use the waterfall model.
- In practice, parts of a phase can be started before next.
- Breakdown a project into phases reduces overall development time.
- However, errors may be found, forcing feedback and backtracking to earlier phases.
- This fits the description of the Spiral model.

Prototypes

- One way to detect errors in requirements early is to build a prototype and give it to users as soon as possible (throw away prototype, start over).
- We can iterate through the requirement, design, implementation and acceptance phases.
- Advantages:
  - Errors in requirements will become obvious once the prototype is in the hands of customer.
- Disadvantages:
  - In complex systems, prototypes costs more to implement. They may be too valuable to throw away.
  - Prototype may be poor basis for final development.

Goals of requirements analysis

- Requirements provide a precise description of the needs of the customer.
- A product may be developed before the customer exists. In that case, a group in the organization plays the role of the customer.
- A customer's description of this needs might not be complete or precise.
- The goal of requirements analysis is to gather complete and precise description.
- Gathering requirements is a skill.
- In these scenario, the customer is king (or at least, thinks he is).

Existing systems

- Usually a new product replaces an existing system.
- This system might be ...
  - ... non-computerized.
  - ... a combination of programs and external process.
  - ... a program with unsatisfactory performance.
- The existing system might give you information on how to do things.
  - Methods for normal processing.
  - How to deal with errors.
  - Integration in organizational context.

Existing systems

- The existing system might give you information on what needs to be done.
  - Missing parts (external process to be incorporated).
  - Unsatisfactory parts which need better performance.
- Requirements analysis considers ...
  - ... normal cases
  - ... user errors (often neglected)

Scenarios

- A scenario is a step-by-step walk-through of an interaction with a system.
- Start with scenarios that capture typical interaction, assuming the user does not make errors.
- Next, consider scenarios that cover user errors.
- If you want to properly document these scenarios, you need to build *Use Cases*.

Users errors

- Errors can be dealt with in a variety of ways:
  - Design the user interface to prevent errors.
    e.g. masks
  - Recognize erroneous input and reject commands.
    e.g. validations
- Some errors can not be detected immediately.
  e.g. A bank clerk makes an error in the amount deposited in an account. This error will not be detected until client examines monthly statement.
  - We need to provide recovery actions as part of requirements.
  - Some Some recovery actions might require external action
    e.g. A check bounces because of bank error. An action is required on the part of bank such as a phone call.

System errors

- Most scenario's describe system responses when the system is working properly.
- This does not help us to define behavior when the system fails.
- The analyst must decided the amount of effort spent to detect and handle software errors.
  - Limit the scope by checking critical modules for reasonable results and shut down on failure.
  - Restart in a clean state.
  - Always log information about failures to preserve information about errors.

Hardware errors

- Analyst must decide what to do and how to prevent hardware failures.
- High availability: system is up and running all the time (failures are infrequent). This requires use of redundant software or hardware
  (e.g. RAID: Redundant Array of Inexpensive Disks).
- High reliability: if the system fails, no information is lost (failures are recoverable).
  (e.g. a daily incremental backup system using tapes stored off-site)
- Availability and reliability are often confused. For example, a RAID does not increase reliability. You still need backups.

Performance requirements

- Performance requirements consist of both space (resource) and time requirements (one is traded off for the other).
- First find out if there are hard limits in either dimension
  - Programs may have to run on a microcomputer with limited memory.
  - Flight controllers may have to calculate altitude of airplane every 1/10th of a second.
- Time requirements may be broken down into two categories:
  - Throughput: amount of data processed in time interval.
  - Response time: time between interaction with system.

Requirements affecting design

- Some requirements are not part of the finished product but do affect the design phase.
- Modifiability: which parts of the system are likely to change in the near future. The design can be shaped to simplify certain changes.
- Reusability: which parts of the system are likely to be reused in future versions or similar systems. The design can be shaped to enable reuse.

Constraints on delivery schedule

- The system may be needed very soon.
- This affects the design by trading off performance for simplicity of implementation.
- Some parts of the system may be needed earlier.
- This affects implementation schedule and design.

Summary

- Functional requirements:
  - How does a correctly functioning program respond to correct and incorrect user interactions?
  - How does the program respond to hardware and software errors?
- Performance requirements:
  - How fast must certain actions perform?
  - What are the constraints on primary and secondary storage?
- Potential modifications:
  - What are likely changes or extensions to the product?
- Delivery schedule:
  - Which parts of the product need to be delivered early?

Midterm Results

- The average was 70%.
- Reading the questions will help you answer the question.
- The booklet has lines, use them.

Question 1.1

What is abstraction? How does abstraction relate to decomposition?

Abstraction is decomposition by changing the level of detail to be considered. It allows us to forget information and consequently to treat things that are different as if they were the same.

Success Rate: 77 %

Question 1.2

What is the difference between an abstract class and an interface?

An abstract class can contain code and can only be inherited once. An interface cannot contain code and can be inherited multiple times.

Success Rate: 77 %

Question 1.3

What is the difference between overloading and overriding?

Overloading allows multiple functions with the same name, but taking different types, to be defined. Overriding allows a class to replace the implementation of method that it has inherited from the super-class.

Success Rate: 77 %

Question 1.4

What is a Java primitive?

Any variable that is not an object.

Success Rate: 73 %

Question 1.5

What is the difference between a total and a partial implementation of a function?

A total implementation of a function will accept any input. The partial implementation of a function will have a REQUIRES clause and will limit the scope of the input that can be given to the function.

Success Rate: 91 %

Question 1.6

What is minimal constraining?

When minimal constraining, specification should constrain details of the procedure only to the extent necessary. In other words, we should impose the minimum number of constrains on the input of a function.

Success Rate: 73 %

Question 1.7

What is the difference between == and the *equals* method?

The == sign is built into the Java language and will compare objects by reference. Equals() is a method defined at the object level and should compare mutable objects by reference and immutable objects by content.

Success Rate: 82 %

Question 1.8

What is the difference between declaring a function public and declaring a function protected?

A protected function is visible only to classes inside the same package and subclasses (even those outside the package). By contrast a public function is visible everywhere and by all subclasses.

Success Rate: 32 %

Question 1.9

Why do most exceptions have the word *Exception* at the end of their name?

It's a convention. It makes it easier to recognize exceptions when reading code.

Success Rate: 45 %

Question 1.10

What is the difference between an iterator and a generator?

The iterator is a procedure that returns a generator object. A generator (implements java.util.Iterator) is an object that produces the elements (that does the enumerations).

The iterator is also an interface in the Java library which defines the behavior of the generator.

Success Rate: 45 %

Question 1.11

Why must Java do method dispatch at runtime?

Because the compiler does not know the real type of an object. It only know the apparent type (which is not enough for method dispatch).

Success Rate: 82 %

Question 1.12

What is the difference between the *Comparable* interface and the *Comparator* interface.

Comparable allows an object to compare itself to another compatible object. This is called the element subtype approach. Comparator is used to compare two separate object with implementing it at the level of the objects. This is called the related subtype approach.

Success Rate: 64 %

Question 1.13

Why would a programmer use the *Runnable* interface instead of extending the *Thread* class?

Java only allows single inheritance. This means that if an object is already extending another object, the programmer must use the Runnable interface instead.

Success Rate: 50 %

Question 1.14

What is the difference between a *Socket* object and a *ServerSocket* object?

The Socket object is the end-point for a TCP/IP communication. The ServerSocket object only listens for new connections and accepts them.

Success Rate: 64 %

Question 1.15

How do you make an object serializable? What method(s) do you need to implement?

You only need to implement the Serializable interface. There are NO methods you NEED to implement.

Success Rate: 50 %

Question 2.1

What will be printed to STDOut?

```
Class A Function A
Class B Function A
Class B Function B
Class B Function A
Class B Function A
Class B Function A
```

You lost 1 point per incorrect answer (maximum of 2).

Success Rate: 70 %

Question 2.2

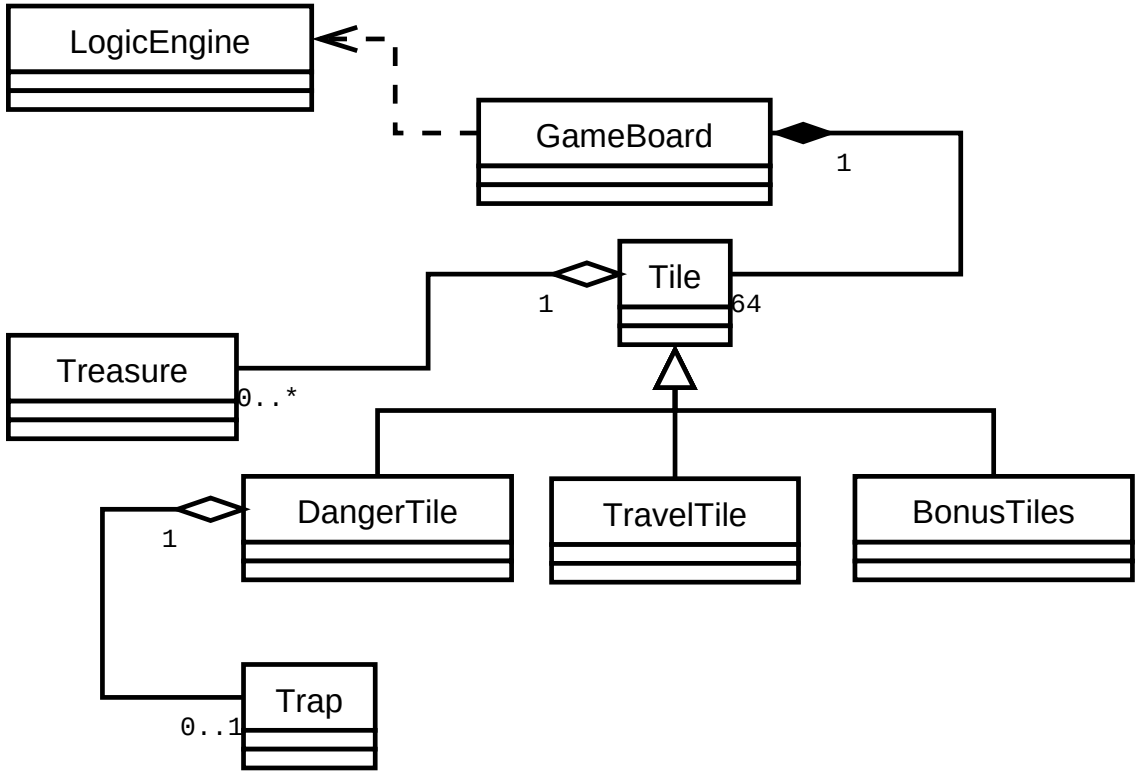
Assuming memory has just been garbage collected and no dead object remains, after the following two statements, how many dead Poly objects does the heap have?

```
The following piece of code will create an object:


new Poly()
sub()
new Poly(5,2)
add()
add()
minus()
add()
new Poly(4,3)
minus

That makes 9 objects. Since there is only one reference, the 8 others
will be dead in the heap.
```

Success Rate: 68 %

Question 3

Draw a simple UML diagram (no methods or properties) to describe the following game software.



You got 1 point if you had less that 3 errors in your diagram.

Success Rate: 73 %

Question 4

Name one *tool of the day* we saw at the end of a class and explain how it is related to Java and/or this class.

All of the following were valid tools of the day:

- Air Conditioners
- CVS
- Nokia 5100
- Dia
- jEdit
- Jikes
- JavaDoc

Success Rate: 95 %

Tool of the day: Jar files

- The JavaTM Archive (JAR) file format enables you to bundle multiple files into a single archive file.
- Creating a Jar file is very similar to creating a Tar file (or Zip).
- The JAR file format provides many benefits :
  - Packaging: You can bundle multiple files into a single.
  - Security: You can digitally sign the contents of a JAR file.
  - Compression: The JAR format allows you to compress your files for efficient storage.
  - Portability: The mechanism for handling JAR files is a standard part of the Java platform's core API.