Exceptions

Guang Song
Q1: What is an exception?
A1:
- improper use of API (violating API), malicious use, programming errors, etc.
- Unavoidable failures due to the system (most IO related failures)
- Q2: why throwing exception?
- A2:
  - Exceptions have two important aspects: report (throwing) and remedy (catching). There is often a distance between the spot where an exception occurs to where the competent handler is located. Java exception mechanism enables the flow of control be immediately transferred from the problem to the remedy.
  - When a method detects a problem it cannot solve, it is better to throw an exception than to try to come up with an imperfect fix.
• Q3: what kind of exception to throw? Checked or unchecked?
Q4: How to define a (checked or unchecked) exception class?
• Q5: how to throw (or generate) an exception?
Q6: what else must be done to the method when you throw an exception in a method? If the exception is checked, or unchecked?
• Q7: what must be done when you invoke a method that throws an exception? If it is checked, or unchecked?
Q8: When to catch an exception?

A8:

- A method should only catch an exception if it can really remedy the situation. Otherwise, the best remedy is simply to have exception propagate to its caller, allowing it to be caught by a competent handler.
Q9: where will the flow of control go when an exception does occur?
• Q10: How to use try-catch block?
• Q11: When to use finally block?
• A11:

  – Use the finally clause whenever you need to do some clean up, such as closing a file or releasing other system resources.

  PrintWriter out = new PrintWriter(filename);
  try {
    for(int i=0; i<steps; ++i) {
      writeData(out, expensiveComputation(i));
    }
  }
  finally {
    out.close();
  }