CSCI-230: Exam II Guideline
Dr. Imad Rahal

YOU ARE ALLOWED ONE CHEAT SHEET (FRONT & BACK)

- *UML*:
  - Use case diagrams, use case descriptions, class diagrams and communication diagrams
  - Draw a use case diagram for a given set of requirements
  - Provide a detailed use case description
  - Given a class diagram, create corresponding Java classes
  - Given a set of related Java classes, devise a corresponding a class diagram
  - Given a class diagram and a use case diagram, create communication diagrams for specific use case scenarios
  - Be able to answer questions on a given UML diagram

- *Linux*
  - Redirecting I/O
  - Pipelining
  - Permissions
  - Shell scripting
  - Linux commands such as **`ls`**, **`cd`**, **`file`**, **`grep`** (with **`–v`** option), **`find`**, **`pine`**, **`sort`**, **`cat`**, **`tar`**, **`zip`**, **`chmod`** etc …

- *CVS*
  - The CVS process
  - How CVS works?
  - The normal sequence of CVS commands
  - All covered CVS commands with their major options and when to use them: **`cvs init`**, **`cvs import`**, **`cvs co`**, **`cvs commit`**, **`cvs update`**, **`cvs add`**, **`cvs rm`**, **`cvs diff`**
  - Understand output returned by CVS commands (e.g. for **`cvs update`**, **`cvs commit`**, etc …)

- *Debugging*
  - Failures, Errors and Faults
  - Syntax vs. logic mistakes
  - Spatial and temporal relationships between failures and faults
  - The debugging process: from failures to errors
  - Breakpoints & debugging commands in Netbeans: **`finish, continue, step over, step over expression, step into, step out`**
  - Given some code:
    - Be able to show runtime stack and stack frames
    - Be able to tell where the debugger goes to next upon issuing a debugging command

- *Testing & JUnit*
  - What is testing?
  - The different types of testing activities and when to use each (Unit, Integration, Functional, etc …)
  - Unit testing: black box and white box testing
    - Generate test cases (input/output combinations) for a given method using black box testing
      - Generate equivalence classes for every input (name & range)

        - Select sample test values per input
        - Include Boundary/special cases
        - Combine test values optimistically or pessimistically
    - Generate test cases (path/input/output combinations) for a given method using white box testing
        - Draw complete and correct flowcharts
        - Create a table showing paths, inputs & outputs
  - Understand code involving **`TestCase`** and **`TestSuite`**
  - Major assert methods used in JUnit: (e.g. **`assertEquals`**, **`assertTrue`**, etc …)
  - Know how to write JUnit test code including testing for Exceptions
  - Failure vs. Error in JUnit

- *Web Programming*
  - Static vs. dynamic Webpages and how a Web server processes each
  - Basic HTML: given HTML code show corresponding Web page (and vice versa)
  - Form processing: method attribute, action attribute, hidden fields
  - Java Servlets: given Servlet code show corresponding Web page (and vice versa)
  - JSP: tags, **`request`** object (**`request.getParameter`** method), **`response`** object(**`response.sendRedirect`** method), **`session`** object (**`session.setAttribute`** & **`session.getAttribute`** method)
  - JSP: given .jsp code show corresponding Web page (and vice versa)
  - Create or complete a JSP page or Servlet to do something specific

- ***Refactoring***
  - Why use and when to use the following refactorings: **Rename**, **Encapsulate Field**, **Move Field/Method**, **Extract Method**, **Pull Up Field/Method**, **Push Down Field/Method**
  - The steps on how to apply the following on a given piece of code (MUST BE ABLE TO FOLLOW THE EXACT SAME STEPS DISCUSSED IN CLASS): **Extract Method** and **Move Method**
  - Code bad smells

- ***Databases** & **JDBC***
  - Understand the following concepts: *Data*, *Database*, *Database Management* System (DBMS), *Database System*, *Primary Key, Entity Integrity*, *Foreign Key*, *Referential Integrity*, *JDBC driver* and *SQL*
  - Write **`Create Table`** SQL statements to create a given table: be able to specify common data types (**`Integer`**, **`Decimal(x,y)`**, **`Char(x)`**, **`Varchar(x)`**) for table fields as well as primary keys and foreign keys
  - Write **`Select`** SQL statements to answer queries
  - JDBC: Be able to write/understand JDBC code involving the following objects
    - **`Connection`** (**`DriverManager.getConnection`**)
    - **`Statement`** (.**`executeQuery`** vs .**`executeUpdate`**)
    - **`ResultSet`**
      - ✓ Process a **`ResultSet`**
      - ✓ Map SQL types to the appropriate Java method (**`getX`** methods)