COMP2100 cheat sheet

Data Structures

- Binary Tree: common algorithms in binary search tree like searching , deleting and traversing:

```
// Traversal questions is common
public List<object> traverse() {

}
public void traverse(Node node, List<object> result) {
    if (node == null) return;
    // preorder traversal
    result.add(node.value);
    traverse(node.left);
    traverse(node.right);

    // inorder traversal
    traverse(node.left);
    traverse(node.right);
    result.add(node.value);

    // postorder traversal
    traverse(node.left);
    traverse(node.right);
    result.add(node.value);
}
```

- RB tree: questions usually about double checking the property of RB tree like:
  - each node is either this color or that color:

```
private boolean test1Helper(Node<T> node) {
    if (node == null) return true;
    return node.colour != Colour.VIOLET && test1Helper(node.left) && test1Helper(nod
e.right);
  }
```

  - root and every leaf are a color using DFS:

```
public Boolean testProp2() {
    //START YOUR CODE
    return root.colour == Colour.PINK && test2Helper(root);

    //END YOUR CODE
  }
  private boolean test2Helper(Node<T> node) {
    if (node.left == null && node.right == null) return node.colour == Colour.PINK;
    boolean result = true;
    if (node.left != null) result = test2Helper(node.left);
    if (node.right != null) result = result && test2Helper(node.right);
    return result;
  }
```

  - and some other properties:
    - Each path from Root-to-Leaf has the same number of PINK nodes.
    - A MAGENTA node cannot have a MAGENTA child, and must at least has a PURPLE child.
    - A PURPLE node must have only PINK children. link to code
- AVL tree (not common): some rotation algorithms and self balance after insertion link to code

Software Testing

```
exampleMethod(int a, int b, int c, int d) {
    if (a > 0 && c == 1) {
```

```
        statementX;
    }
    if (b == 3 || d < 0) {
        statementY;
    }
}
```

- Branch Complete (**very common**): check all possible branches (go through all if and else blocks) link to pdf

```
someMethod(true, true, true);
someMethod(false, true, true);
someMethod(false, false, false);
```

Not common:

- Path Complete: Check all possible paths within the control flow graph. Link to PDF
- Statement Complete: Check all possible statements (code complete maybe). Link to PDF
- Condition Complete: Check if all conditional statements (i.e., conditions in if) have been evaluated to both true and false (not common). Link to PDF
- Multiple Condition Complete: Test all possible combinations of conditions. Link to PDF

Parser & Tokeniser

- Tokenizer: Mostly implement the `next()` function with pattern matching. Link to code
    - Use `startsWith(Token.Type.enum)` to return the right token.

```
if (_buffer.startsWith(Token.Type.LEFT.toString()))
            return new Token(Token.Type.LEFT);
```

    - Use `substring(index start, index end)` to choose the right value for the Token object.

```
_buffer.substring(0, _buffer.indexOf(";"));
```

  - Other helpful functions: `Character.isDigit()`, `Character.isLetter()`, `"str".equalsIgnoreCase()`.

- Parser: Implement the grammar. Usually, the grammar is simple and can be carried out using recursive calls or while loops to form an object or return a value. Link to code

  - recursive (more common):

```
public void parseExp() {
    if (_tokenizer.hasNext()) parseCommand();
    if (_tokenizer.hasNext()) parseExp();
  }
public void parseCommand() {
    Token token = _tokenizer.takeNext();
    // doing change based on the grammar
  }
```

  - While loop:

```
public void parseExp() {
    Token insert = tokeniser.takeNext();
    // grammar checking here
    while (tokeniser.hasNext()) {
      Token values = tokeniser.takeNext();
      // logic code from the output from tokeniser
    }

  }
```

Persistent Data

- Saving and loading XML file locally. There are a lot of boiler plate code

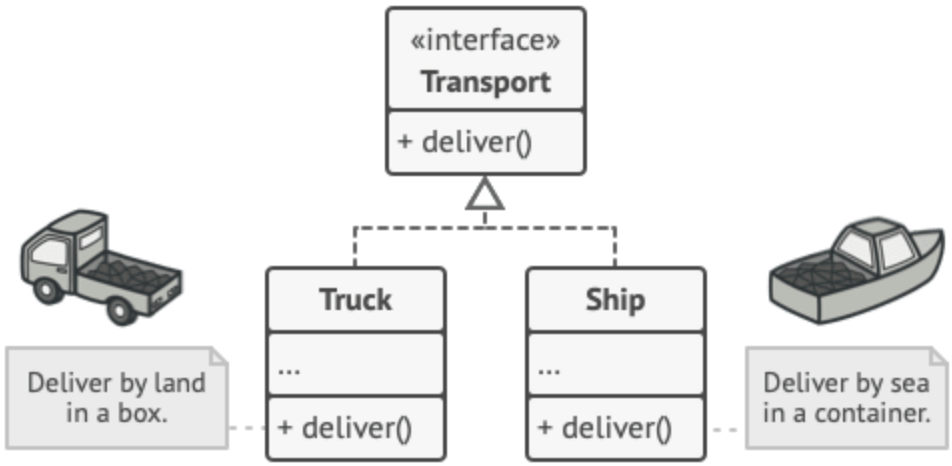
  - saving usually a list of object as XML file.
  - read csv file locally, simpler than saving.

Design Patterns

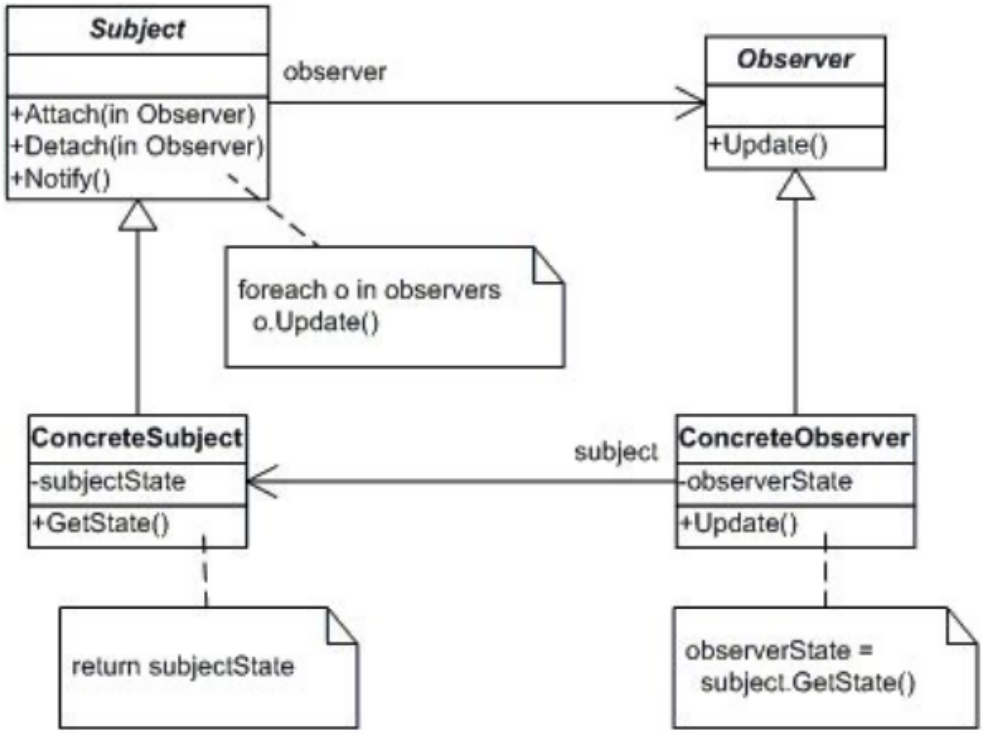
- Singleton: ensures one global instance of a class.

```
public class Example {
    private static Example instance;
    private Example() {}
    public static Example getInstance() {
        if (instance == null) instance = new Example();
        return instance;
    }
}
```

- Factory Method: selects which object to instantiate based on the subclass. Link to code



- Observer: establishes a one-to-many relationship between objects, so when one object changes its state, all dependent objects will be notified and perform an action. Link to code



- State (difficult): changes the behavior of an object as its state changes. Link to code
- Template Method: defines the skeleton of an algorithm in an operation, allowing subclasses to redefine certain steps of an algorithm without changing its structure. Link to code
- Iterator (easy): abstracts the data traversing process. Link to code