COMP160 Cheat Sheet

by exo via cheatography.com/19823/cs/2776/

Lecture 14 - References

* Reference types include String, Array and other classes.
* **This** is a keyword that always refers to the current instance object.
* No need for different parameter names.

Lecture 16 - Arrays 2

* Arrays are objects and can hold references to objects.
* The state of an array can be changed by a reference to an array (a change to the reference b for example will also change a)
* An array of classes can be made just as if you were creating an array of primative values: `AClass [] z = {new AClass(), new AClass(3.3, "hello", 3), new AClass() };`

Lecture 18 - events

* Objects that represent events in the GUI (button clicks, mosue clicks, text input etc.)
* Remember to import java.awt.*; javax.swing.*; java.awt.event.*;
* Event model: (event generator object makes an) Programmer creates components and registers listener for each one. (even object reference sent to) automatically handlered whe user generates an event. (event listener object) programmer creates listener objects

Lecture 21 - Hierarchies, Inheritance

* Extends: we can extend a class to create/define a new class (subclass/child) A child class has it own member (datafields & members) and also non private members of the parent class.
* Hierarchy: A class inherits from its parent, this is how objects get methods like toString that aren't its own
* Super: keyword refering to members of current class/object inherited from the parent class

Lecture 24 - Collections, ArrayList, Applets

* Enumerated Types: Enumerated types are a way of creating a new type by enumerating (listing) every possible value.
* Generics: where programs are written in terms of "to be specified later".
* Collections: Java has many ways of representing and processing collections of objects, using classes and interfaces in java.util. These are called collection classes, collection framework, or collection API.
* ArrayList: A data structure which can store varying numbers of (references to) objects in a flexible list. import java.util.ArrayList. Doesn't hold primitive types.

```
* ArrayList<String> a1 = new
ArrayList <String>();
a1.add("red");
a1.add("green");
String a = a1.get(1);
system.out.println(a);
a.indexOf("yellow");
```

Lecture 15 - Arrays 1

* An array is a named collection of data of same type and a fixed size.
* Index can only be an int (or a char cast to an int)
* Hold primitive types, references / handles / pointers to objects
* Initialiser lists: `int []intAr = { 3, 7, 9, 23, 2 };`
* An array always starts at position 0 and goes to length -1
* Multidimensional Arrays - `int [][]a = new int [3] [5];` initializing = `double [][]b = { {1.0, 3.4}, {5.6, 8.9} };`

for-each

```
for(double [ ] row : b ) {
  for(double x : row ) {
     System.out.print( x + " " );
  }
  System.out.println();
}
```

Listener

```
public void actionPerformed
(ActionEvent event) {
  if (event.getSource() == left)
    label.setText( "Left" );
  else
    label.setText( "Right" );
}
```

Lecture 22 - Visibility, overriding

* Packages: A package is a related group of classes. Ones that you don't put into a package get put into the default package.
* Visibility - package visibility is used when no visibility is specified - accessible only to classes in this package. protected visibility - accessible to classes in this package and any class extending this one
* Method overloading: A class can have multiple methods of the same name as long as they have different parameter specifications.
* Method Overriding: methods with same signature (name and parameter specification) exist in both parent and child class.
* When datafields have the same name it is called shadowing

ArrayList

`ArrayList()`	Constructor: creates an initially empty list
`void add (Object obj)`	Inserts the specified object to the end of this list
`void add(int index, Object object)`	Inserts the specified element at the specified position in this list
`void clear()`	removes all of the elements from this list
`Object remove (int index)`	Removes the element at the specified position in this list
`Object get (int index)`	Returns the element at the specified position in this list
`int indexOf(Object obj)`	

ArrayList (cont)

Returns the index of the first occurrence of the specified element in this list,
or -1 if this list does not contain the element

`boolean contains(Object obj)`

Returns true if this list contains the specified element

`boolean isEmpty`

Returns true if this list contains no elements

`int size()`

Returns the number of elements in this list

Printing out a 2D array

```
for(int row = 0; row < b.length;
row++) {
  for(int col = 0; col <
b[row].length; col++){
    System.out.print( b[row] [col ]
+ " ");
  }
System.out.println();
}
```

Lecture 17 - Graphics components

* **Components**: Objects that are drawn in the GUI (including containers such as frames and panels).
* A Container is a type of component that holds and organises other components JPanel holds other components, JFrame holds JPanels in a window
* **Layout**: (default) FlowLayout: Left to right top to bottom. BorderLayout: 5 positions, north south east west centre. BoxLayout: Single row or column. GridLayout: a grid. Example:
`add(b5.BorderLayout.East)`
*

Lecture 20 - Input/Output

* A stream is a source or destination of a sequence of data.
* Tokens are text items separated by "white space"
* try..catch: exception handling. when error occurs: do nothing - program will crash, handle the error with try catch.
* Use a scanner to read from a file.

Lecture 23 - Abstract classes

* instanceof (boolean): returns true if an object is an instance of a class i.e. s intanceof Square
* Abstract Classes: They are used when you want to create instances of subclasses but not the abstract parent itself. Abstract class can have its own normal methods.
* Abstract methods must be implemented in child classes, have no body in parent class.
* A child class can only have one parent. Multiple inheritance.
* Java has interfaces to avoid full multiple inheritance. Interfaces may only contain abstract methods and static final data fields.
* Final: a final class may not be extended, a final method may not be over ridden.