Scope, Pass-by-Value, Static

- The golden rule for static methods to know is that static methods can only modify static variables.
- Declaration Instantiation (= Assignment)
- Declaration - 64 bit in global frame, Instantiation - memory space, Assignment copies the bits

	Static variable	Static method	Instance variable	Instance method
Type used by compiler	Static type	Static type	Static type	Static type
Type used by executer	Static type	Static type	Static type	Dynamic type

- Instance variable lookup: start from *static type* class. If not there, compile error.
- Instance method lookup: start from *dynamic type* class. If not there, go to *superclass* until *Object* class is reached.
- Variables aren't polymorphic. Only *instance* methods are.

- == determines identity while .equals is for equality
- Static v. Non-Static
  - Non-static members cannot be invoked using class name
  - Non-static method, aka Instance method (if the method is going to be invoked by an instance of the class, then it's non-static). If the method needs to use self (this) instance vars, then the method must be non-static
  - Use "private" keyword to prevent code in other classes from using members (or constructors) of a class
  - Use "static" keyword if the nested class never uses any instance vars or methods of the outer class (minor savings of memory)
- 8 primitive types: byte, short, int 32 bits, long, float, double 64 bits, boolean, char + 9th reference to Objects (including String)
- Golden Rule of Equals (GRoE) and Parameter Passing: the equals sign copies the bits, pass by value (e.g. b = a)

Linked Lists, Arrays, Iterators, Exceptions

- Arrays: Declaration, Instantiation, Assignment
  - int[] a = new int[3]; – initiated 3 slots with default 0's
  - int[] x = new int[] {0, 2, 3, 10};
  - int[] y = {1, 2, 3}; Cannot do x = {1, 2}; i.e. this instantiation can only be used with declaration.
- Use .length for arrays, and .length() for Strings.
- System.arraycopy(source_arr, sourcePos, dest_arr, destPos, len);
- int[][] matrix = new int[4][4]; <— creates (1+4=)5 array
- Generic Lists - parameterize the input using reference type (Capitalize),
  i.e. Item[] items = (Item[]) new Object[100]; when creating an array of references to "Items"
- Iterators
  - An Iterable implements: Iterable<Type>,
    then it must define public Iterator<Type> iterator()
  - An Iterator implements Iterator<Type> must have two methods: (boolean) hasNext, and (Type) next
- int a; a++; causes an error since a isn't instantiated.

Inheritance

- You can only call *one* other constructor from a constructor, and the call *has* to be on the first line. This call is "super" which means the superclass' constructor. You can use "this(...)" to call a different constructor defined in the same class.)
  - ((String) var).method(); casts var to (String) and then apply the method
  - (String) var.method(); casts the return of this method to (String)
- @Override (good habit).
- General flow for one argument methods, suppose we have a.call(b): [ST = Static type, DT = dynamic type].
  1. During compile time, java only cares about static types. First, check if a's ST has a method that takes in the ST of b.
     (a) If not, check a's superclasses for a method that takes in ST of b.
     (b) If not, check if any of the methods take in supertype of ST of b, as we are looking for b's "is-a" relationships. Start from a's ST methods and move up from its superclass.
     (c) If still not, Compile-Error!
  2. Take a snapshot of the method found, i.e. java memorizes its signature.
     (a) The method signature that is chosen at runtime will try to exactly match with our snapshot. The signature consists of the method name, and the number and type of its paramaters.
  3. During runtime, if call is an overriden method, then run a's dynamic type's call method. If call is an overloaded method, then run the most specific snapshot.
  4. Runtime errors can consist of downcasting (as seen in Corgi c3 = (Corgi) new Dog();), but also many that are not related to inheritance (NullPoint- erException, IndexOutOfBound-Exception, etc).
- Encounter linked-lists, consider sentinelFirst and sentinelLast; Encounter array-lists, consider looping around.
- Interfaces cannot be instantiated; An interface's fields (vars) must be labelled public final
- Overloading - methods with the same name (diff. param type) - bad
- Interface - all subclasses must implement all its methods (except possibly default methods)
- Overriding - subclasses must override all methods; has to have the exact same signature (from return type, name, to params, etc.) and only counts for non-static methods
- Rules
  - Compiler allows memory box to hold any subtype
  - Compiler allows calls based on static type
  - Overriden non-static methods are selected at run time based on dynamic type. Everything else is based on static type, including overloaded methods.
  - (Compiler only considers compile-time type, but actual behavior is based on dynamic type.)

Extra Sanity Checks

- Manually run through test cases to check
- N.B. Care for null base cases; null out unreferenced boxes
- Check functionalities of all provided methods as helper
- Curly braces around statements, and semicolons ; after each line
- Before Java variables can be used, they must be declared.
- Java vars must have a specific type, which can never change.
- public static void main(String[] args)
  - true/false are NOT capitalized.
  - && and, || or.
  - if () {} else if () {} else {}
- Default values
  - int: 0 (float: 0.0)
  - boolean: false
  - String (any reference type): null
- Subclass constructors ALWAYS calls a super constructor first (by default the empty param one is provided). You can only call *one* other constructor from a constructor, and the call *has* to be on the first line. This call is "super" which means the superclass' constructor. You can use "this(...)" to call a different constructor defined in the same class.)
- super.super.xx is NOT allowed.
- Consider base cases like if (x.rest == null) / if (x == null)
- Don't mix up == and .equals (the former compares the bits, only works for null and numbers)
- JUnit @Test
  - import org.junit.Test; import static org.junit.Assert.*;
  - assertArrayEquals, assertEquals, assertNotEquals(expected, actual), assertTrue, assertFalse
- str1.toUpperCase(); and str2.toLowerCase();
- Step Into does one step at a time and enters method calls if necessary; Step Over completes the entire method call (if there is one) and continues to the next line of the current program.
- Deque<Integer> sth = new ArrayDeque<>(); −okay, but Deque<Integer> sth = new Deque<>(); since Deque is an interface
- Set<String> S = new HashSet<>();
- Exceptions: throw new xxxException("xxxx");
- Subtype Polymorphism provides a single interface to entities of different types