Sun Certified Programmer for Java 2 exam

Initialization

- All class-level (member) attributes (static/non-static) are initialized by default.
- Local (automatic) variables are not initialized by default. They must be done explicitly. Failure do to so is a compiler error.
- However, an array local variable's contents are initialized by default – to 0's of nulls (not the actual reference itself but the contents of the array).
- Final variables must be initialized when they are declared.
- Member initializion has problems with exceptions:
    - Cannot call methods that throw checked exceptions.
    - Cannot do catches for checked exceptions.
    - Can have static blocks for static attributes.
    - Can have initialization block instead of ctors, eg:
      Class A { long[] s = new long[2]; { s[0]=1L;s[1]=2L; }
    - Obviously the more recognizable place is in ctors.

Strings

- Strings are immutable.
- There are methods that look like they'd change the String but in fact they create a new one and return it, eg: concat, trim, replace.
- There are lots of 'valueOf' methods that parse primitives.
- Each primitive's wrapper classes has a 'parseXXX' operation that does the reverse of above, returns a Wrapper object, and throws a NumberFormatException.
- In substring(int,int) the parameters are start and end point, not length.
- In indexOf method it returns –1 if not found.
- The JVM has a string pool where it keeps at most one object of any String. String literals always refer to an object in the string pool. String objects created with the new operator do not refer to objects in the string pool but can be made to using String's intern() method. Two String references to 'equal' strings in the string pool will be '=='.
- If there is a "abc" + "def" the compiler will optimize it to "abcdef", so "abcdef" == "abc" + "def" is true.
- StringBuffer doesn't override equals.

Arrays

- Arrays are objects. The following create a reference for an int array:
    - int[] ii;
    - int ii[];
- You can create an array object with new or an explicit initializer:
    - ii = new int[3];
    - ii = new int[] { 1,2,3 };
    - int[] ii = { 1,2,3); // only when you declare the reference.
- CAREFUL: You can't create an array object with:
    - int iA[3];
- If you don't provides values, the elements of obj arrays are always initialized to null and those of primitive arrays are always initialized to 0.
- If you don't provide the values, elements of characher arrays are initialised to blanks. Array of all other primitive datatypes is initialised to 0 or 0.0 depending upon whether the datatype is int or float.
- The elements wont initialize unless the size is specified. e.g. if the char (or of any primitive datatype) array is initialised as done in class below:

```
class test1
{
      int i,j,k;
```

```
        char[] c ;
}
```

The class would compile clean but throw a runtime exception (Null Pointer Exception) wherever an attempt is made to print values of c[0], c[1].

Primitive Types

- Primitive types:

Primitive	***Number of bytes***	***Min/max***
Byte	1 (8 bits)	-128 -> 127
Short	2 (16 bits)	-32768 -> 32767
Int	4 (32 bits)	`-2147483648 -> -2147483647`
Long	8 (64 bits)	big
Float	4 (32 bits)	Not needed
Double	8 (64 bits)	Not needed
Char	2 (16 bits – Unicode chars)	0 ->

- Literals:
  - You can have boolean, char, int, long, float, double and String literals.
  - You cannot have byte or short literals.
- char literals: 'd', '\u0c20' (the 0c20 **must be a 4-digit hex number**).
- int literals: 0x3c0 is hex, 010 is octal(for 8).
- You can initialize byte, short and char variables with int literals (or const int expressions) provided the int is in the appropriate range.
- CAREFUL: can't assign a double literal to a float, eg: float fff = 26.55;
- The only bit operators allowed for booleans are &^| (cant do ~ or shift ops)
- Primitive wrapper classes are immutable, override equals, the static valueOf(String) methods in primitive wrapper classes return wrapper objects rather than a primitives and throws NumberFormatException.

Conversions and Promotions

- Byte -> short –> int -> long -> float -> double.
  char -> int " "
- All other primitive conversions are allowed with an explicit cast.
- char/byte/short/int/long -> float/double is a widening conversion even if some precision is lost (the overall magnitude is always preserved).
- Narrowing conversions require an explicit cast.
- integral narrowing conversions simply discard high-order bits.
- anything->char is a narrowing conversion (inc byte) because its signed->unsigned and negative numbers get messed up, eg: int x = 32; char y = (char)x ; - is in fact now a space.
- Widening primitive and reference conversions are allowed for assignment and in matching the arguments to a method (or ctor) call.
- For assignment (but not method invocation), representable constant int expressions can be converted to byte, char or shorts (eg. char c = 65), eg: final int C = 33 ; char y = C ; - now a '!'
- Unary numeric operators implicit promotion: byte/short/char -> int, eg: byte x = 3; x = (byte)-b ;
- Binary numeric promotions:
  - both arguments are made (in order of preference) double/float/long/int.
  - include (in)equality operators.
  - eg: short n = 40; n = (short)2 + n ;
- char/byte/short are promoted to int for nearly every operator. Be careful not to assign the uncast int return value to a narrower type, eg:
  - bytesum = byte1 + byte2; // won't compile without a cast.
  - bytesum += byte2; // is ok.
- applies to bitshift operators (as a unary promotion on each arg).
- there are no promotions for ++, --, += etc.

- A switch argument can be any type that can implicit-cast to an int (byte/char/short/int but not boolean or long).
- The argument and cases can also be compile-time constant expressions.
- CAREFULL: using a short in a switch - the cases must also be short even though it has been promoted to int!
- Explicit Casting:
    - Impossible casts are detected at compile time.
    - Other bad casts cause runtime exceptions rather than messes.
- Array casts:
    - The only implicit conversion for arrays is: Base[] base = new Der[5];
    - a runtime exception is thrown if you try to add anything but Derived
- There are no implicit casts for arrays of primitives – they aren't polymorphic
- You can make an explicit array cast: String[] strs = (String[]) ObjectArray;

Bitshift operators

- int i = 12 << 4 ; // 21 shift 4 times to the left, 192.
- The left-hand argument to a bitshift operator can be an int, long or any type that can be implicit cast to int (byte,char,short).
- char, byte, or short arguments will be promoted to int before the shift takes place, and the result will be an int (so has to be cast to get back to the original type).
- You can't shift further than the number of bits in the left-hand argument (int or long). Only the five (six) low-order bits of the right-hand argument will be used to shift an int (long).
- Each left-shift (signed-right-shift) for positive numbers is equivalent to multiplication (division) by 2. The division is rounded down.
- >>> does not preserve the sign bit, >> does.
- >>> on a positive number is the same as >>.

Object-oriented concepts

- The signature of a method is its name, argument type and argument order.
- Overload = loads of the same method
- an overload is legal provided the two methods have different signatures.
- an override must have the same signature and return type, throw no new checked exceptions and be at least as accessible as the method it is overriding.
- An override method can widen the scope of the method, eg: protected -> public.
- An override method can narrow the exceptions being thrown.
- Fields do not act polymorphically:
    - Whenever you access a field from outside the class, the declared type of the reference is used rather than the type of the object it refers to.
    - Regardless of the type of the reference, a method will always access its own fields rather than those of a derived class.
- CAUTION: Private method calls are statically bound (no virtual table). A method will call the private method in the class where it is defined even if the calling method is inherited by a derived class and the derived class defines a method with the same signature as the private method.
- Calls to public and protected methods in the same class are dynamically bound even for ctors (and from private methods). This is different to C++.
- Re-using a static method's name in a derived class:
    - A static method cannot be overridden by a non-static method
    - A static method can be overridden by a static method but does not dynamically bind (therefore static methods can't be abstract)
    - You can overload a static method with a non-static method.
- note also that a static method can be inherited.
- note that a static var can be 'overriden' by a non-static var.
- It's legal to assign any reference type including an Interface reference to an Object reference which makes sense because the interface ref must point to an Object (or null).

Nested classes

- A nested class is a class that is defined inside another class.
- There are two distinct types of nested classes:
    - static nested classes (or top-level nested classes)
    - inner classes (which are always associated with an instance of an enclosing class), eg: member classes, local classes (in a code block) and anonymous classes.
- Top-level classes, eg: static nested classes and package member classes.
- Access to enclosing class:
    - all outer-class members (inc. private) are accessible to an inner class (Usually without scope modifiers, but if they are hidden by an inner class name, you can use Outer.this.outerVar)
    - static nested classes cannot access instance variables from enclosing classes (there is no instance), but can access their static variables.
- Instantiation:
    - For accessible inner classes: Outer.Inner i = new Outer().new Inner();
    - Even if you are in Outer's scope, you need to have an Outer instance.
    - For accessible static nested classes: Outer.Nested nested = new Outer.Nested();
- Local inner classes cannot access non-final local variables or **method** arguments.
- Nested classes generally have the same options with regard to modifiers as do variables declared in the same place. Note: you cannot use access modifiers for local classes (in block of code).
- Unlike class-level nested classes, local classes are executed in the method's sequence of execution so you can't create an instance of the local class before it is declared.
- The static keyword marks a top-level construct (class, method or field) and can never be subject to an enclosing instance.
    - no inner class can have a static member.
    - no method can have a static member.
- Interfaces automatically attach 'public static final' to field and class members (thus making them top-level nested rather than member inner classes).
- A nested class cannot have the same simple name as any of its enclosing classes (note: there is no similar restriction on method or variable names).
- The compiler creates:
  Outerclass.class for the outside class
  Outerclass$Innerclass.class for an inner class
  Outerclass$1.class for an anonymous class inside Outerclass.Innerclas – surprising eh?

Threads

- Know where the basic thread methods are:
    - wait, notify and notifyAll are Object instance methods.
    - start, stop, suspend, resume and interrupt are Thread instance methods.
    - sleep and yield are Thread static methods
- Thread states: Bruce Eckel lists 4 thread states: new, runnable, blocked, dead.
- Blocking
    - There are 5 ways a thread can be blocked - sleep, wait, suspend, synchronization, io blocking.
    - sleep and suspend do not release locks held by the thread.
- Deprecated methods
    - stop is unsafe because it releases all locks and may leave objects in an inconsistent state.
    - suspend is deprecated because its failure to release locks makes it prone to deadlock. Calling wait in a sync block is safer.
- The isAlive method returns false for new threads as well as dead threads.
- Threads inherit their priority from the thread that calls their ctor.
- Object.wait can also take a time in milliseconds so how long it will wait for – don't get confused with Thread.sleep which won't wake up with a notify and doesn't loose the monitor.
- Priorities 1 – 10, or Thread.MIN_PRIORITY(1), MAX_PRIORITY(10) and NORM_PRIORITY (5)

Exceptions

- Non-runtime exceptions are called checked exceptions. Unchecked being those that inherit from RuntimeException or Error.

- Even if a method explicitly throws a runtime exception, there is no obligation for the caller to acknowledge the exception. One consequence of this is that the restriction on exceptions thrown by an overriding method only applies to checked exceptions.
- A try block's finally clause is called unless the JVM is exited – I think?
    - a return in try or catch does not prevent finally from being executed.
- A try block's finally statement will be executed (unless the thread dies) before control leaves the try/catch/finally scope. It will be executed before unhandled exceptions (from try or catch) are passed back up the calling stack.
- If you return from a try and finally does not return. 1) the return value is calculated, 2) finally executes and 3) the method returns with the value calculated prior to executing finally.
- If you have a return in both try and finally, the finally's value is always returned.
- Primitive floating point operations do not throw exceptions. They use NaN and infinity instead.
- A ctor can throw any exception.
- The order of catches checked by compiler, eg: you must catch FileNotFoundException before IOException.

Streams

- System.in is an InputStream and out and err are PrintStreams.
- Beware of instances of the abstract OutputStream and InputStream.
- OutputStreams:
    - write(int) writes the eight low-order bits to the underlying stream
    - write(byte[]) write(byte[],int off,int len)
    - flush()
    - close()
- BufferedOutputStream has two ctors: (OutputStream) (OutpuStream, int size)
- DataOutputStream writes primitives and strings to a byte-based stream.
- ObjectOutputStream writes primitives, strings and serializable objects (inc arrays) and is not a FilterOutputStream (seems strange).
- PrintStream
    - two ctors: (OutputStream) (OutputStream, boolean autoflush)
    - never throws IOExceptions (sets internal flag instead)
    - print and println for primitives and strings.
    - print(Object) prints Object.toString().
    - System.out and System.err are printstreams.
- FileOutputStream has 4 constructors: (File) (FileDescriptor) (String) and (String, boolean append)
- PipedOutputStream has two ctors: () and (PipedInputStream)
    - method to connect(PipedInputStream)
- ByteArrayOutputStream has 2 ctors: () and (int size)
- Writers:
    - OutputStreamWriter
    - StringWriter, CharArrayWriter - like ByteArrayOutputStream
    - FileWriter, BufferedWriter, PrintWriter, FilterWriter
    - There is no ObjectWriter, DataWriter instead use PrintWriter.
    - PrintWriter can be constructed with an OutputStream or Writer.
- Readers:
    - LineNumberReader doesn't attach numbers, it has getLineNumber().
    - CharArrayReader is constructed with the char[] that it reads from and optional int start position and length args.
- RandomAccessFile
    - does not extend File or any type of stream.
    - two ctors: (File or String name, String mode="r" or "rw")
    - checks for read/write access at construction (unlike File).
    - has read/write methods for primitives and strings (and a couple of others).
    - has a (byte) file-pointer: seek(long), long length().
- FileDescriptor
    - just a handle to a sink/source of bytes.
    - only has two methods: sync() and valid()
- File
    - represents an abstract file or dir pathname (file/dir doesn’t have to exist).

    - 3 ctors: (File or String parent, String child) (String pathname)
    - methods to delete, renameTo and lots to specify a filename in a O/S independent way.

Collections

- The Collection interface is extended by Set and List
    - a set contains no duplicates.
    - a list is sequenced and can have duplicates.
- The Map interface does not extend Collection and is extended by SortedMap.
- There are abstract classes for each of the main interfaces (Collection, Set, List and Map) and implementations extend these.
- Set implementations: HashSet (unordered/unsynchronised), TreeSet (ordered/unsynch)
- List implementations: ArrayList (ordered/unsynchronised), LinkedList (head/tail ops/unsynch), Vector (ordered/synch), Stack(LIFO/synch).
- Map implementations: HashMap(allows nulls/unsync), TreeMap (sorted/unsync), Hashtable (no nulls/synch)
- There are Collections and Arrays classes to provide static methods for general algorithms on collections and arrays.

Math methods

- The java.lang.Math class not java.math package!
- most act on and return double values
- note that floor(double), ceil(double) and rint(double) return doubles not ints
- abs, max and min can take double, float, int or long arguments and the return type matches the argument type. Note: abs(Integer.MIN_VALUE) returns a negative because the positive can't be represented as the primitive – same applies for other primitives.
- There are three rounding methods:
    - int round(float); // note both are 32-bit
    - long round(double); // note both are 64-bit
    - double rint(double); // returns double though
- log returns natural log.
- trig functions (eg: sin, cos, tan, asin, acos, atan) take double arguments in radians. Note: Math.toRadians method.

AWT

- Component is abstract. Container is not abstract by extends it.
- Checkbox has 5 ctors: no-arg + four with String as first arg and combinations of boolean initialState and CheckboxGroup.
- CheckboxMenuItem has nothing to do with Checkbox or CheckboxGroup.
- Events – semantic (Action/Item/Text/Adjustment), low level (the rest – those that inherit from ComponentEvent).
- InputEvent (super for MouseEvent and KeyEvent) has a 'long getWhen()' method.
- MouseEvent has 'int getX', 'int getY' and 'Point getPoint' methods.
- ComponentEvent has a 'getComponent' method to get the ref of the component that the event was for. Semantic events aren't component events so they have to use getSource() and cast.
- Semantic events have only one method on their listener IF so don't have adaptors.
- Listeners:
- KeyListener: Pressed/Released/Typed
- FocusListener: Lost/Gained
- ComponentListener: Shown/Hidden Moved Resized
- ContainerListener: component- Added/Removed
- WindowListener: Opened/Closing/Closed, Activated/Deactivated (keyboard focus) and Iconifed/Deiconified
- MouseListener: Pressed/Released/Clicked, Entered/Exited and MouseMotionListener: Dragged/Moved
- ActionListener – actionPerformed.
- Who generates what:

- ActionEvent – Button, List, MenuItem and TextField.
- ItemEvent – Checkbox, CheckboxMenuItem, Choice and List.
- TextEvent – TextField and TextArea.
- AdjustmentEvent - Scrollbar

Layout

- BorderLayout is the default layout for Window/Frame/Dialog.
  - add(component, BorderLayout.NORTH) == add("North", component)
  - adding a component without an explicit position is identical to adding a component at BorderLayout.CENTER.
  - if more than one component is added to the same position, the most recently added component is displayed there.
  - north,south,east and west components only expand on one axis.
  - It Is a capital 'i' shaped.
- FlowLayout is the default layout for Panels (including Applets).
  - if the panel is bigger than its components, they are centred horizontally and at the top (ie. north position).
- GridLayout with container larger than components expands to fill its container provided that the number of components matches the rows*columns.
  - Empty rows are given space, while empty rows are not.
  - If there aren't enough components, it will try to fill its rows first.
- Ctors for Border-, Flow- and GridLayout can take int hgap and vgap args.
- FlowLayout(int align, hgap, vgap)
- Be careful with nested layout questions. eg Button->panel->frame, the panel fills the whole frame, but the button will be its preferred size at north position in the panel (and thus the frame).
- GridBagLayout
  - There are two ways to set the constraints for a component in a gbl:
    - container.add(Component c, Object gbc)
    - gbl.setConstraints(Component c, GridBagConstraints gbc)
  - weightx and weighty are doubles (0->maxDouble, default 0) and determine where extra width or height is added if a row or column is smaller than the container.
  - gridwidth and gridheight are ints (1->maxInt) ... RELATIVE, REMAINDER.
  - gridx and gridy are ints (0->maxInt) ... RELATIVE
  - RELATIVE can apply to gridx/gridy (=next) or gridwidth/gridheight (=second last)
- A Component added to a null layout won't display unless you set its bounds. This is the only place where a component can control its parent's layout directly.

Miscellaneous

- Illegal arguments (eg. adding a container's parent to the container, adding a window) get through the compiler but throw an exception at runtime.
- A class type's name is valid as an identifier, eg. int Boolean = 5;
- Modifiers:
  - Transient fields are not written out when a class is serialized.
  - Transient and Volatile can only be applied to fields.
  - Native can only be applied to methods.
  - You can have a static synchronized method. It is synchronized on the class object not the 'this' object.
  - static transient is legal but doesn't effect anything.
- The right-hand-side of an assignment can be a reference to null.
- You can println a null reference (and add it to another string).
- What you can't do with a null reference is, nullRef.aMethod();
- Order of Operation
  - arithmetic before &^|
  - & then ^ then |
- Garbage Collection
  - unreachable objects can become reachable (if their finalize method causes another object to have a reference to them).

    - objects referenced by block-level variables are not available for garbage collection until their enclosing method's scope is exited.
- The compiler never object to creating a local instance of the class being constructed in its constructor. If the call produces an infinite loop, a runtime error will occur.
- Labelled break and continue statements:
    - The label referred to by a labelled break statement is attached to a statement block, which could be a loop or switch but doesn't have to be.
    - The label referred to by a labelled continue statement must be attached to a loop.

Things to look out for

- Read the answers before going through code samples.
- LOOK FOR NON-STATIC METHODS/VARS ACCESSED FROM MAIN
- WHEN YOU FINISH THE EXAM GO BACK AND CHECK FOR NON-STATIC METHODS/VARS FROM MAIN AND OTHER STATIC METHODS
- if (...)
  statement1;
  statement2; //...always executed
- Beware of an otherwise legal override having a more restrictive access modifier.
- Beware of missing return statements.
- Look for static modifiers and make sure they dont contain refs to instance vars.
- Beware of standard methods (eg. main, paint, run) with the wrong args or return type.
- With array declarations, look out for "int intArray[5] = ..."
- Beware of adding a primitive to a Vector.
    - more generally, you can't use a primitive where an Object is required (eg. the equals method).
- System.out.println(aReferenceToNull); // is fine
- Beware of local vars from a try block used in its catch block (out of scope).