• Write code that declares, constructs, and initializes arrays of any base type using any of the permitted forms both for declaration and for initialization.

Creating an Array requires 3 things:

Declaration:

int[] iA; Integer tA[]; Student[] sA; Size is never specified in declaration.

Allocation:

iA = new int[10]; You are allocating 10 places for storing 10 integers.

sA = new Student[5]; You are allocating 5 places for storing 5 references to Student Objects. Note: Not 5 places to store Student objects!

Size must always be specified in allocation.

Initialization:

Remember, when you allocate an array space is allocated only to store the references. In case of primitive types the space contains 0 (for integral types), 0.0 (for floating points) or false( for booleans). In other cases, these places are initialized to point to 'null'.

2 ways to explicitly initialize arrays:

First way:

for (int i = 0; i < iA.length; i++) iA[i] = 5;

for (int i = 0; i < sA.length; i++) sA[i] = new Student();

Second way:  (Initialization the time of allocation.)

iA = new int[] {5, 5, 5, 5, 5, 5, 5, 5, 5, 5 };

sA = new Student[] { new Student(), new Student(), new Student(), new Student(), new Student() };

The above statements are creating an int array of 10 elements and a Student array of 5 elements respectively. Note that, the size is not specified explicitly.

Multidimensional Arrays:

Java supports Arrays of Arrays. This means you can have something like:

int[][] iAA; iAA is an array. And each of it's elements points to an object which is an array of ints.

It can be instantiated like this:

int[][] iAA = new int[2][3]; ie. An array containing 2 elements. Each element in turn points to another array of ints containing 3 elements.

Rules for multidimensional array are exactly same as single dimensional array. Eg. in the above case, the 2 places of iAA are pointing to null.

Or iAA = new int[][]{ {1, 2, 3}, null }; Here, first element of iAA is pointing to an array containing {1, 2, 3} and the second element is pointing to null.

Array features:

Array size is always maintained with the array object as a final instance variable named 'length'.
Ex. iAA.length . You cannot assign any value to it. ie. you cannot do iAA.length = 4;


Array Pitfalls:
An Array is a first class java object. Eg. Assuming int[] iA == new int[3];  (iA instance of Object) is true.
When you create an array of primitive type eg. int[] iA == new int[3]; , you create 1 object and three primitives initialized to 0. But when you create an array of any java object, eg.  String[] sA == new String[3]; , you create 1 object and 3 places initially pointing to 'null' to hold 3 string objects.

Difference between
  int[] a, b;  // a and b both are arrays
  and int a[], b; // a is an array but b is just an int.
Indexing starts with 0. ie. First element is accessed as: iA[0] = 10;


_____




• Declare classes, inner classes, methods, instance variables, static variables, and automatic (method local) variables making appropriate use of all permitted modifiers (such as public, final,static, abstract, and so forth). State the significance of each of these modifiers both singly and in combination, and state the effect of package relationships on declared items qualified by these modifiers.
There are 5 kind of classes:
1. The standard/normal one. This is called " TopLevel class ".
2. Static Member Class or (previously known as "Top Level Nested Class"):  A static class defined inside a class. Just like static methods, it can only access static members of the outer class.
3. Inner class: An inner class is a nested class that is not explicitly or implicitly declared static. Inner classes cannot declare static initializers or member interfaces. Inner classes cannot declare static members, unless they are compile-time constant fields eg. public final static int CODE = 100; //this is valid inside an inner class.
4. Local Class: A local class is a nested class that is not a member of any class and that has a name. All local classes are inner classes. Every local class declaration statement is immediately contained by a block. Local class declaration statements may be intermixed freely with other kinds of statements in the block.The scope of a local class declared in a block is the rest of the immediately enclosing block, including its own class declaration. Local class cannot have any of these modifiers:: public, protected, private, or static.
5 Anonymous Class: It is a class that does not have a name. There are only 2 ways to create such classes:
SomeClass sc = new SomeClass()

```
{
  public void m1()
  {
  }
};
```
SomeClass is an existing class. So, sc refers to an object of 'anonymous' class which is a subclass of SomeClass.
```
SomeInterface sc = new SomeInterface()
{
 public void m1() { } //implement all methods of SomeInterface.
};
```
SomeInterface is an existing interface. So, sc refers to an object of 'anonymous' class which implements SomeInterface.

• An anonymous class is never abstract or static.

• An anonymous class is always an inner class.

• An anonymous class is always implicitly final.

Points to remember:

• A nested class is any class whose declaration occurs within the body of another class or interface.

• A top level class is a class that is not a nested class.

• A member class is a class whose declaration is directly enclosed in another class or interface declaration.

• Similarly, a member interface is an interface whose declaration is directly enclosed in another class or interface declaration.

• Member interfaces are always implicitly static so they are never considered to be inner classes.

• A Class : CAN BE public, final, abstract CANNOT BE synchronized, native, private, protected, static, volatile

• Members of a class:

. Fields : CAN BE public, private, protected, static, volatile, final CANNOT BE synchronized, native, abstract

. Methods : CAN BE public, private, protected, static, native, synchronized, final, abstract CANNOT BE volatile

. Fields inside a method are called 'automatic' or 'local' variables. Such fields CANNOT BE public, private, protected, synchronized, native, abstract, static, volatile

You need to have a very good understanding of access levels (what private, protected, public means and how it works). Read this topic from any good java book.

Few points to remember about private, protected, public (also known as access modifiers):

• Only applied to class(only public can be applied to top level class declaration) and class members (constructors, fields or methods).

• Fields inside methods cannot have these modifiers and they are visible only inside that method.

• Members may be inaccesible even if the class is accessible. It depends on members' access modifier. But if the class is not accessible, the members will not be accessible, even if they are

declared public.

• If no access modifier is specified, ie. default access, then it means it is only visible in that package ie. all classes in the same package can access it.

• "private" means only that class can access the member. Also, "private" means, private to a class not to an object ie. An instance of a class can access the private features of another instance of the same class.

• "protected" means all classes in the same package (like default) and sub-classes in any package can access it.

• static members cannot access non-static members. But non-static members can access static members.

• Order of restrictiveness: (Least Restrictive) public < protected < default (no modifier) < private (Most Restrictive)

_____

• For a given class, determine if a default constructor will be created, and if so, state the prototype of that constructor.

Default constructor will be created ONLY when a class does not define any constructor explicitly.
For eg
public class A
{
 public A() //This constructor is automatically inserted by the compiler as there is no other constructor defined by the programmer explicitly.
 {
  super(); //Note this. It is calling super(). This means the super class should have a constructor that takes no parameters. Otherwise this class won't compile.
 }
}

public class A
{
 //Compiler will not generate any constructor as programmer has defined a constructor. public A(int i)
 {
   //do something
 }
}

The access modifier of the default constructor (provided by the compiler, not the one that you write) is same as that of the class. In the top case, it is public because the class is public.

See a detailed example here. Another related example.

_____

• State the legal return types for any method given the declarations of all related methods in this or parent classes.

First, very imp. points:
• A method signature includes: name of the method, no. of parameters, class/type of parameters.
• A method signature DOES NOT include: return type, any other modifier(abstract, final, static etc), throws clause.
Eg. If you say two methods have same signature, that means, their name, no. or params and class/type of params are same. NOTHING CAN BE SAID ABOUT ANYTHING ELSE.
Two cases that you need to consider:
1. Overriding: A subclass has a method with same signature as a method in super class. This means, subclass method is overriding the superclass method. OR super class method is overridden by the subclass method.
Rules:
• Overriding method's return type MUST be the same as that of overridden method's return type.
• Overriding method's throws clause must be compatible. Ie. it can throw any of the exceptions declared in the throws clause of parentclass's (overridden) method, it can throw any SubClass of execptions defined in parentclass's (overridden) method, it may choose not to throw ANY EXCEPTION. In applying the above rule, only CHECKED (ie. exceptions NOT extending from RuntimeException class) are to be considered.
• ANY METHOD CAN THROW UnChecked exception without declaring.
2. Overloading : A class having more than 1 methods with same name but different parameter list. (Note that this makes their signature different). There are no rules. In affect, overloaded methods are entirely independent of each other.Remember, if signature is different, it is a different method.

_____