Declaration and Access Control

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

- An array is a collection of items. It can contain any type of element value (primitive types or objects), but you can't store different types in a single array.
- To create an array in Java, you use three steps:

1.Declare a variable to hold the array.
2.Create a new array object and assign it to the array variable.
3.Store things in that array.

- The simplest form of array declaration is,

```
int a[];

(or)

int[] a;
```

- To declare an array of 2 dimension,

```
int a[][];
or
int []a[];
or
int[][] a;
```

- Array variables indicate the type of object the array will hold and the name of the array, followed by empty brackets ([]). The square brackets can come after the data type or after the variable name
- The array dimension is not specified in a declaration like the above. An array's size is set when you assign something to it.
- The declaration of an array just gets a variable that can hold a reference to an array and does not allocate any storage.
- There are two ways for Creating an array object and assigning it to a variable.
- Using new
- Directly initializing the contents of that array

The first way is to use the new operator to create a new instance of an array:

```
int[] a = new int[10];
```

- When you create a new array object using new, you must indicate how many elements that array will hold. The above example creates an array that can hold 10 integers.
- When you create an array object using new, all its elements are initialized for you (0 for numeric arrays, false for boolean, '\0' for character arrays, and null for objects).
- To create an array and initialize its contents at the same time, instead of using new to create the new array object, enclose the elements of the array inside braces, separated by commas:

```
String[] names = { "Harry", "Jo", "John", "Jim", "Diana" };

int[][] i ={{1,2,3},{10,9,8}};
```

- The size of the number of elements you've included will be automatically created for you. This example creates an array of String objects named "names" that contains five elements.

- To get the size of an array:

```
int len = names.length;
```

variable len will have 5, as there are 5 elements in the names array.

- The above example has 5 elements which are accessed using subscript 0 to 4

```
System.out.println("Name: " + names[1]);
```

will print "Jo" and *not* "Harry"

```
System.out.println("Name: " + names[5]);
```

will give you a Runtime Exception called ArrayIndexOutOfBoundsException

- To assign a value to a particular array element,

```
names[1] = "Jack";
```

Remember: The length of the array is 5, but its subscript can only go up to 4. Arrays start numbering from 0. Subtract 1 from the length of the array to get its largest element.

---

2) 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.

public :

- class is visible in other packages.
- field is visible everywhere (class must be public too)

private :

- Private variables or methods may be used only by an instance of the same class that declares the variable or method
- A private feature may only be accessed by the class that owns the feature.

protected :

- Is available to all classes in the same package and also available to all subclasses of the class that owns the protected feature.
- This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.

default :

- What you get by default ie, without any access modifier (ie, public private or protected).
- It means that it is visible to all within a particular package

abstract :

- abstract class must be extended/subclassed (to be useful). It serves as a template.

- A class that is abstract may not be instantiated (ie, you may not call its constructor)
- abstract class may contain static data.
- Any class with an abstract method is automatically abstract itself, and must be declared as such.
- A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated.

```
eg,
public abstract class A {
public abstract void print(); //Note :No curley braces
//{} and no method body defined
public void test() {
//testing.....
}
}

class B extends A {
public void print() {
System.out.println("hi");
}
}
```

- An abstract method has no body. its purpose is to force a subclass to override it and provide concrete implementation of it.
- A compile−time error occurs if a method declaration that contains the keyword abstract also contains any one of the keywords private, static, final, native or synchronized

static :

- static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.
- static methods are implicitly final, because overriding is done based on the type of the object, and static methods are attached to a class, not an object.
  A static method in a superclass can be shadowed by another static method in a subclass, as long as the original method was not declared final. However, you can't override a static method with a nonstatic method. In other words, you can't change a static method into an instance method in a subclass.

```
public class test extends atest{
test() { }
public void method() { }
public static void main(String arg[]) {
}
}
class atest {
public static void method(){ }
}
```

  You will get a compiler error saying "The instance method void method() declared in class test cannot override the static method of the same signature declared in class atest. It is illegal to override a static method."
- static methods can only directly access other static field.
- static method has no "this".

```
public class test {
public static void method() {
this.print();
}
public static void print() {
System.out.println("Test");
}
public static void main(String arg[]) {
method();
}
}
```

The class fails to compile stating that the variable "this" is undefined.

- With ordinary non−static data and methods you must create an object and use that object to access the data or method, since non−static data and methods must know the particular object they are working with.
- Since static methods don't need any objects to be created before they are used, they cannot directly access non−static members or methods by simply calling those other members without referring to a named object

```
eg,

class stat {
static int x = 0;
stat(){x++;}
}

stat s1 = new stat();
stat s2 = new stat();
s1.x=10;
s2.x=20;
y=s1.x;
```

y will have 20 and not 10 because x is static. So the better way to refer a static variable is via class name stat.x=200;

way to access static field inside a static method is,

```
class stat {
static int x = 0;
public static void main(String s[]) {
System.out.println(x);
}
}
```

way to access non−static field inside a static method is,

```
class stat {
int x = 0;
```

```
public static void main(String s[]) {
stat s = new stat();
System.out.println(s.x);
}
}
```

final :

- class can't be extended ie., final class may not be subclassed.
- final method can't be overridden when its class is inherited.
- can't change value of a final variable (is a constant)

```
class subMath extends java.lang.Math {
}
```

You will get a compiler error saying "Can't subclass final classes" because java.lang.Math class is final.

- A final method cannot be overridden.

```
class Test {
final void print() {}
}
class anotherTest extends Test {
void print() {} //final methods can't be overridden
}
```

- A final variable is a constant.

eg for static and final variable :

```
class stat {
static int x=5;
int z = 5;
final int y = 6;
public static void main(String arg[]) {
stat s = new stat();
s.x=7;
s.z=9;
s.y=8; //can't assign a value to final variable
System.out.println(s.x); //prints 7
System.out.println(stat.x); //prints 7
System.out.println(stat.z); //can't make static
//reference to non−static variable
System.out.println(s.z); //prints 9
}
}
```

modifiers	*class*	*methods*	*variables*
final	Y	Y	Y

abstract	Y	Y	N
static	N	Y	Y
native	N	Y	N
transient	N	N	Y
volatile	N	N	Y
synchronized	N	Y	N

Remember :

- abstract and final can't be used together.
- A constructor can only be qualified with public, private and protected.
- Don't make any fields public without good reason.

	Member Visibility			
Accessible to	***public***	***protected***	***default***	***private***
same class	Y	Y	Y	Y
class in same package	Y	Y	Y	N
subclass in different package	Y	Y	N	N
Non−subclass in different package	Y	N	N	N

---

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

Constructors look a lot like regular methods, but some of them to be noted are :

- Constructors always have the same name as the class.
- Constructors don't have a return type.

```
class aClass{
String s;
void aClass() {
s="hello";
}
void print() {
System.out.println(s);
}
public static void main(String arg[]) {
aClass a = new aClass();
a.print();
```

```
}
}
```

output :
null

As there is a return type, "void aClass()" is considered as a method and not as a constructor. So, variable s is local to method aClass() and not accessible in method print()

- Constructors may take one or more (or even no) parameters.
- Constructors are always called with the *new* keyword.
- You can have more than one constructor in a class.
- Constructor is generally available for a class only if it is explicitly defined in that class. The exception is the "default constructor". The default constructor takes no arguements and is created by the compiler if no other constructors are defined for the class.

Note: The defualt constructor is not inherited. It is created for you by the compiler if and only if, you don't provide any other constructors in the source of the particular class.

eg,

```
class Construct{
Construct(int i) {
intvalue=i;
}
public static void main(String str[]) {
Construct c = new Construct();
}
}
```

This gives error saying there is no matching constructor. if you didn't create any constructor explicitly, creating this instance with new won't give any error, because it takes the default constructor.

- To call a constructor defined in the superclass, you have to place the call as the first statement in the calling constructor.

---

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

See objective 19
