Advanced Class Design

**Inheritance**

* Derived class constructor must call its base class constructor. A subclass must call at least one constructor of superclass (implicitly or explicitly)

**Abstract classes and methods**

* Abstract method cannot have body
* Must be implemented in derived non-abstract class
* Abstract class does not need to have abstract methods
* Abstract method cannot be defined in concrete class
* Cannot be instantiated (equals to concrete class with all private constructors)
* Abstract class can extend another abstract or concrete class
* Cannot define abstract static method
* Abstract class cannot be instantiated and class marked as final cannot be extended, therefore following code will not compile:

abstract final class Animal {}

* Abstract class doesn’t have to implement all methods from interface it implements. It can leave this to concrete class that will extend it



**Static keyword**

* You can define variables, methods, nested classes, and nested interfaces as static members.
* Top-level classes, interfaces, and enums can’t be defined as static entities

**Static variables**

* Unlike instance variables, static variables are initialized only once when they are loaded into memory. Default values are:
  + False for Boolean
  + \u000 for char
  + 0 for byte, short, int, long, float, double
  + null for objects
* combination of **static and final modifiers is used to create constants**

**Static Initializer blocks**

* Instance variables can be instantiated using constructors or initializer blocks.
* Static variables can be instantiated using Static initializer blocks.
* Multiple blocks possible, execute in order of appearance
* All types of assignment are possible – declaration, initialization, assignment, calling other static methods
* Since you cannot throw checked exception in static method, you can call static method from static initializer block and catch the exception.

Execute in order of appearance, hence it finishes with compilation error “cannot forward reference”

class DemoMultipleStaticBlocks {

static {

++staticVar;

}

static int staticVar ;

static {

++staticVar;

}

public DemoMultipleStaticBlocks() {

System.out.println("Constructor:" + staticVar);

}

public static void main(String args[]) {

new DemoMultipleStaticBlocks();

}

}

**Static methods**

* can access static variables and other static methods , cannot access instance variables or call instance methods. Opposite is possible.

**Final keyword**

* final class – you don’t want it subclassed
* final method – you don’t want it overridden
* final variable – you don’t want it changed after initialization

**Final Variables**

* any variable (static, instance, local , method parameter) can be tagged final
* static final variable can be initialized with declaration or in static initialization block
* instance final variable can be initialized with declaration or in initialization block
* if not used only local final variable (variable in a method) can survive compilation uninitialized.
* final variable must be initialized in class it is declared in. Cannot be initialized in derived class. Therefore, following code will not compile.

abstract class MyClass {

public final int finalVar;

}

class MyDerivedClass extends MyClass {

MyDerivedClass() {

super();

finalVar = 1000;

}

}

**Final Methods**

* cannot be overridden
* The private methods of a base class aren’t inherited by its derived classes. A method using the same signature in the derived class isn’t an overridding method, but a new method.

class Base {

private final void finalMethod() {}

}

class Derived extends Base {

final void finalMethod() {}

}

**Final Class**

* Cannot declare final and abstract class.
* Cannot be extended
* Interface cannot be final, because its abstract by default

**ENUM**

* Implicitly defined as final entity
* Implicitly extends class Enum
* For above two reasons cannot be subclassed
* All enum constants are implicitly declared – final, static, public. Constants are initialized in static initializer block. Hence constructors fire first.
* Method values() returns List of constants
* Method valueOf() returns enum constant for corresponding String value
* Method ordinal() return position of constant in Enum
* Default order of enum constants is the order they are created in
* Methods name() and toString() return the variables name. toString() can be overridden, while name() is final
* Enum constant list must be the first in enum definition, otherwise you get compilation error
* Semicolon at the end of constant list is optional if you don’t add methods to Enum
* You can also define constructors in your enum and override methods that apply only to particular enum constants (IceCream3.java)
* An enum constant can define a constant specific class body and use it to override existing methods or define new variables and methods. If constant specific method is not overriding method from Enum constant creates anonymous class.
* Can be declared as top-level enum, member of class or interface
* Enum can define abstract method, but needs to be overridden for each constant