1- Difference between inner class and lambda

Anonymous Inner Class	Lambda Expression
It is a class without name.	It is a method without name.(anonymous function)
It can extend abstract and concrete class.	It can't extend abstract and concrete class.
It can implement an interface that contains any number of abstract methods.	It can implement an interface which contains a single abstract methods.
Inside this we can declare instance variables.	It does not allow declaration of instance variables, whether the variables declared simply act as local variables.
Anonymous inner class can be instantiated.	Lambda Expression can't be instantiated.
Inside Anonymous inner class, "this" always refers to current anonymous inner class object but not to outer object.	Inside Lambda Expression, "this" always refers to current outer class object that is, enclosing class object.
It is the best choice if we want to handle multiple methods.	It is the best choice if we want to handle interface.
At the time of compilation, a separate .class file will be generated.	At the time of compilation, no separate .class file will be generated. It simply convert it into private method outer class.
Memory allocation is on demand, whenever we are creating an object.	It resides in a permanent memory of JVM.

2- Difference between thread and process

Anonymous Inner Class	Lambda Expression
An executing instance of a program is called a process.	A thread is a subset of the process.
Some operating systems use the term ‘task‘ to refer to a program that is being executed.	It is termed as a ‘lightweight process’, since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel.
A process is always stored in the main memory also termed as the primary memory or random access memory.	Usually, a process has only one thread of control – one set of machine instructions executing at a time.
Therefore, a process is termed as an active entity. It disappears if the machine is rebooted.	A process may also be made up of multiple threads of execution that execute instructions concurrently.
Several process may be associated with a same program.	Multiple threads of control can exploit the true parallelism possible on multiprocessor systems.
On a multiprocessor system, multiple processes can be executed in parallel.	Multiple threads of control can exploit the true parallelism possible on multiprocessor systems.
On a uni-processor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency.	All the threads running within a process share the same address space, file descriptors, stack and other process related attributes.
Example: Executing multiple instances of the ‘Calculator’ program. Each of the instances are termed as a process.	Since the threads of a process share the same memory, synchronizing the access to the shared data within the process gains unprecedented importance.

3- What Is Type Erasure?

- Type erasure can be explained as the process of enforcing type constraints only at compile time and discarding the element type information at runtime.
- For example:

```
public static  <E> boolean containsElement(E [] elements, E element){
    for (E e : elements){
        if(e.equals(element)){
            return true;
        }
    }
    return false;
}
```

- The compiler replaces the unbound type E with an actual type of Object:

```
public static  boolean containsElement(Object [] elements, Object
element){
    for (Object e : elements){
        if(e.equals(element)){
            return true;
        }
    }
    return false;
}
```

- Therefore, the compiler ensures type safety of our code and prevents runtime errors.
- Class Type Erasure
  - At the class level, the compiler discards the type parameters on the class and replaces them with its first bound, or Object if the type parameter is unbound.
- Method Type Erasure
  - For method-level type erasure, the method's type parameter is not stored but rather converted to its parent type Object if it's unbound or its first bound class when it's bound.
- Bridge Methods
  - The compiler sometimes creates a bridge method. This is a synthetic method created by the Java compiler while compiling a class or interface that extends a parameterized class or implements a parameterized interface where method signatures may be slightly different or ambiguous.

4- Maven dependencyManagement vs. dependencies Tags

- In general, we use the dependencyManagement tag to avoid repeating the version and scope tags when we define our dependencies in the dependencies tag. In this way, the required dependency is declared in a central POM file.
- dependencyManagement
  - This tag consists of a dependencies tag which itself might contain multiple dependency tags. Each dependency is supposed to have at least three main tags: groupId, artifactId, and version.
- dependencies
  - This tag contains a list of dependency tags. Each dependency is supposed to have at least two main tags, which are groupId and artifactId.
- The version and scope tags can be inherited implicitly if we have used the dependencyManagement tag before in the POM file.
- In fact, we usually define the dependencyManagement tag once, preceding the dependencies tag. This is used in order to declare the dependencies in the POM file. This declaration is just an announcement, and it doesn't really add the dependency to the project.
  - We define the version in the dependencyManagement tag, and then we can use the mentioned version without specifying it in the next dependencies tag.
  - dependencyManagement is just a declaration, and it does not really add a dependency. The declared dependencies in this section must be later used by the dependencies tag. It is just the dependencies tag that causes real dependency to happen.