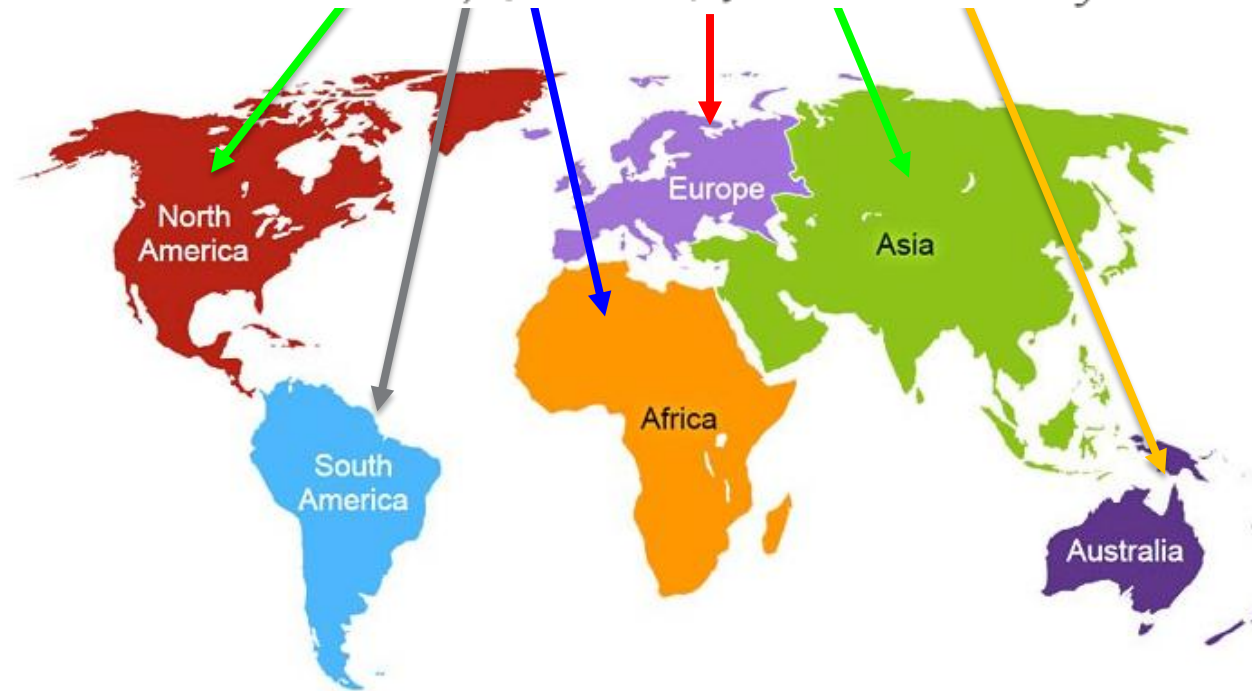





H2K Infosys, LLC
Dream, Strive & Achieve Victory



H2K Infosys is  business based in Atlanta, Georgia – United States
Providing Online IT training services world wide.

www.H2KINFOSYS.com

USA - +1-(770)-777-1269, UK - (020) 3371 7615
Training@H2KInfosys.com / H2KInfosys@Gmail.com

H2K INFOSYS PROVIDES WORLD CLASS SERVICES IN

**IT Trainings
with Real time
Project Work
for
Corporates
&
Individuals**

**Special IT
Training
for MS
Students in
US**

**Software Design,
Development,
QA – Manual,
Automation &
Performance
Testing,
Maintenance.**

**IT Staff
Augmentation

Job Placement
Assistance

Tech Support**

- An *abstract* class is a Class declared as abstract.
- An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon)

If a class includes abstract methods, the class itself must be declared abstract, as in:

```
public abstract class Product{  
    // declare fields  
    // declare non-abstract methods  
    abstract void calculatePackSize();  
}
```

When an abstract class is sub-classed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, the subclass must also be declared abstract.

Unlike interfaces, abstract classes can contain fields that are not static and final, and they can contain implemented methods

If an abstract class contains only abstract method declarations, it should be declared as an interface instead.

When an Abstract Class Implements an Interface, abstract class is not abide to give implementation to all methods of interface.

An abstract class may have static fields and static methods. You can use these static members with a class reference—for example, `AbstractClass.staticMethod()`—as you would with any other class.

- Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on.
- If the conversion goes the other way, this is called unboxing.

```
List<Integer> li = new ArrayList<Integer>();  
for (int i = 1; i < 50; i += 2)  
    li.add(i);
```

The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that expects an object of the corresponding wrapper class.
- Assigned to a variable of the corresponding wrapper class.

The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

- Concurrency is the ability to run several parts of a program or several programs in parallel. If time consuming tasks can be performed asynchronously or in parallel, this improve the throughput and the interactivity of your program.

Process vs. threads

Process: runs independently and isolated of other processes. It cannot directly access shared data in other processes. The resources of the process are allocated to it via the operating system, e.g. memory and CPU time.

Threads: so called lightweight processes which have their own call stack but can access shared data. Every thread has its own memory cache. If a thread reads shared data it stores this data in its own memory cache.

A Java program runs in its own process and by default in one thread. Java supports threads as part of the Java language via the *Thread* code. The Java application can create new threads via this class.

- Java provides locks to protect certain parts of the coding to be executed by several threads at the same time. The simplest way of locking a certain method or Java class is to define the method or class with the *synchronized* keyword. .

The **synchronized** keyword in Java ensures

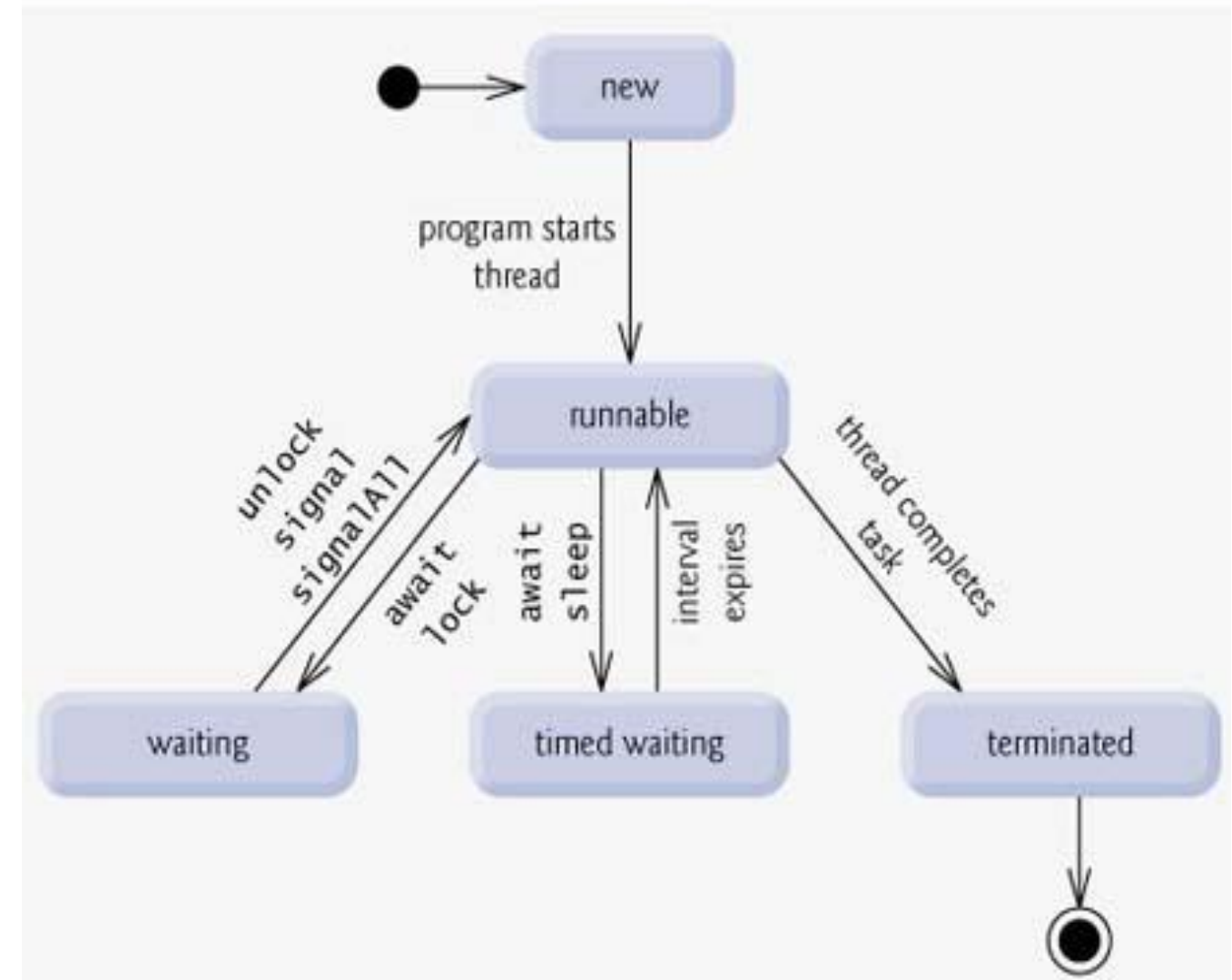
- that only a single thread can execute a block of code at the same time
- that each thread entering a synchronized block of code sees the effects of all previous modifications that were guarded by the same lock

You can use the **synchronized** keyword for the definition of a method. This would ensure that only one thread can enter this method at the same time. Another threads which is calling this method would wait until the first threads leaves this method.

If a variable is declared with the **volatile** keyword then it is guaranteed that any thread that reads the field will see the most recently written value. The volatile keyword will not perform any mutual exclusive lock on the variable.

Threads in Java.

- The base means for concurrency are is the `java.lang.Thread` class. A Thread executes an object of type `java.lang.Runnable`.
- Runnable is an interface with defines the `run()` method. This method is called by the Thread object and contains the work which should be done. Therefore the "Runnable" is the task to perform. The Thread is the worker who is doing this task.



Thread Methods

Method	Action
<code>public void start()</code>	Starts the thread in a separate path of execution, then invokes the <code>run()</code> method on this Thread object
<code>public void run()</code>	If this Thread object was instantiated using a separate Runnable target, the <code>run()</code> method is invoked on that Runnable object.
<code>public final void setName(String name)</code>	Changes the name of the Thread object. There is also a <code>getName()</code> method for retrieving the name.
<code>public final void setPriority(int priority)</code>	Sets the priority of this Thread object. The possible values are between 1 and 10.
<code>public final void setDaemon(boolean on)</code>	A parameter of true denotes this Thread as a daemon thread
<code>public final void join(long millisec)</code>	The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
<code>public void interrupt()</code>	Interrupts this thread, causing it to continue execution if it was blocked for any reason.
<code>public final boolean isAlive()</code>	Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Thread Methods

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread

Method	Action
public static void yield()	Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled
public static void sleep(long millisec)	Causes the currently running thread to block for at least the specified number of milliseconds
public static boolean holdsLock(Object x)	Returns true if the current thread holds the lock on the given Object.
public static Thread currentThread()	Returns a reference to the currently running thread, which is the thread that invokes this method.
public static void dumpStack()	Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

Using the Thread class directly has the following disadvantages.

- Creating a new thread causes some performance overhead
- Too many threads can lead to reduced performance, as the CPU needs to switch between these threads.
- You cannot easily control the number of threads, therefore you may run into out of memory errors due to too many threads.

The `java.util.concurrent` package offers improved support for concurrency compared to the direct usage of `Threads`.

Threads pools with the Executor Framework.

- Thread pools manage a pool of worker threads. The thread pools contains a work queue which holds tasks waiting to get executed.
- In Executor framework: *Executors.newFixedThreadPool(int n)* which will create n worker threads. The *ExecutorService* adds lifecycle methods to the *Executor*, which allows to shutdown the *Executor* and to wait for termination.
- If you want to use one thread pool with one thread which executes several *Runnable*s you can use the *Executors.newSingleThreadExecutor()* method.

```
public class TestClass {  
    private static int NTHREDS = 10;  
    public static void main(String[] args) {  
        ExecutorService executor = Executors.newFixedThreadPool(NTHREDS);  
        for(int i= 10; i < 20; i++){  
            ThreadTask task = new ThreadTask("Thread_" + i, i);  
            executor.execute(task);  
        }  
    }  
}
```

Thank you

Your feedback is highly important to improve our course material and teaching methodologies. Please email your suggestions.

**USA +1-(770)-777-1269
Training@H2KInfosys.com**

**UK (020) 3371 7615
H2KInfosys@Gmail.com**

H2K Infosys is e-Verified business based in Atlanta, Georgia – United States

H2K Infosys acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in this document.



H2K Infosys, LLC (hereinafter “H2K”) acknowledges the proprietary rights of the trademarks and product names of other companies mentioned in any of the training material including but not limited to the handouts, written material, videos, power point presentations, etc. All such training materials are provided to H2K students for learning purposes only. H2K students shall not use such materials for their private gain nor can they sell any such materials to a third party. Some of the examples provided in any such training materials may not be owned by H2K and as such H2K does not claim any proprietary rights for the same. H2K does not guarantee nor is it responsible for such products and projects. H2K acknowledges that any such information or product that has been lawfully received from any third party source is free from restriction and without any breach or violation of law whatsoever.