Difference between "implements Runnable" and "extends Thread" in java

MARCH 12, 2013 LOKESH 8 COMMENTS

Difference between "**implements Runnable**" and "**extends Thread**" in java

In java language, as we all know that there are two ways to create threads. One using Runnable interface and another by extending Thread class.

```
1  public class DemoRunnable implements Runnable {
2      public void run() {
3          //Code
```

```
    }
}
//with a "new Thread(demoRunnable).start()" call

public class DemoThread extends Thread {
    public DemoThread() {
        super("DemoThread");
    }
    public void run() {
        //Code
    }
}
//with a "demoThread.start()" call
```

There has been a good amount of debate on which is better way. Well, I also tried to find out and below is my learning:

1) Implementing Runnable is the preferred way to do it. Here, you're not really specializing or modifying the thread's behavior. You're just giving the thread something to run. That means composition is the better way to go.

2) Java only supports single inheritance, so you can only extend one class.

3) Instantiating an interface gives a cleaner separation between your code and the implementation of threads.

4) Implementing Runnable makes your class more flexible. If you extend thread then the action you're doing is always going to be in a thread. However, if you extend Runnable it doesn't have to be. You can run it in a thread, or pass it to some kind of executor service, or just pass it around as a task within a single threaded application.

5) By extending Thread, each of your threads has a unique object associated with it, whereas implementing Runnable, many threads can share the same runnable instance.

6) If you are working on JDK 4 or lesser, then there is bug :

http://bugs.sun.com/bugdatabase/view_bug.do;jsessionid=5869e03fee226ffffffffc40d4fa881a86e3:WuuT?bug_id=4533087

YOU MAY ALSO LIKE

Inter-thread communication using piped streams in java

Difference between sleep() and wait()?

Difference between yield() and join() in

It's fixed in Java 1.5 but Sun doesn't intend to fix it in 1.4.

The issue is that at construction time, a Thread is added to a list of references in an internal thread table. It won't get removed from that list until its start() method has completed. As long as that reference is there, it won't get garbage collected.

Happy Learning !!

Related Posts:

1. **Inter-thread communication using piped streams in java**
2. **Difference between sleep() and wait()?**
3. **Difference between yield() and join() in threads in java?**
4. **Java executor framework tutorial and best practices**
5. **Writing a deadlock and resolving in java**
6. **Thread synchronization, object level locking and class level locking**

MULTI THREADING | RUNNABLE | THREAD | THREAD IN JAVA

threads in java?

Java executor framework tutorial and best practices

Writing a deadlock and resolving in java

Thread synchronization, object level locking and class level locking

PREVIOUS POST

Some java programmers jokes

NEXT POST

Java executor framework tutorial and best practices

8 THOUGHTS ON “DIFFERENCE BETWEEN “IMPLEMENTS RUNNABLE” AND “EXTENDS THREAD” IN JAVA”

sreenath

SEPTEMBER 3, 2014 AT 7:04 AM

Awesome explanation !!
Small correction in point 4

It should be ” implements Runnable” Not “extend Runnable”

REPLY

pratik

SEPTEMBER 16, 2014 AT 9:32 AM

i want what the main difference in the thread & and Runnable interface?

REPLY

Manoj

DECEMBER 24, 2013 AT 4:57 PM

5) By extending Thread, each of your threads has a unique object associated with it, whereas implementing Runnable, many threads can share the same runnable instance. – This statement seems wrong to me.

Many threads can share the same Thread object, same as Runnable instance.

For example,
public class A1 implements Runnable {….}
public class B1 extends Thread {….}

public class ABMain {
public static void main(String[] args) {
A1 a = new A1(); //Create runnable instance
B1 b = new B1(); // Create Thread instance

new Thread(a).start(); //Starts runnable instance

new Thread(b).start(); //Starts Thread instance.

new Thread(a).start(); // 2nd thread sharing same runnable instance A1
new Thread(b).start(); // 2nd thread sharing same Thread instance.

}
}

Difference between Thread and Runnable I found is, Thread class provides apis to configure Thread instance based on your need.
If you extends Thread class, then
1. you will have flexibility to set priority to Thread instance,
2. you can create daemon thread,
3. you can check if thread is terminated, alive etc.

REPLY

Manoj

DECEMBER 24, 2013 AT 4:59 PM

There are apis available in Thread class which is not avaialbe in Runnable. Those apis helps developers to configure Thread properties.

REPLY

★ **Lokesh Gupta**

DECEMBER 24, 2013 AT 7:45 PM

I modified your program a little bit, to correct the picture of usage of thread and runnable.

```
class A1 implements Runnable {
private int counter = 0;
@Override
public void run() {
counter++;
System.out.println("Running A1 -> " + counter);
}
}

class B1 extends Thread {
private int counter = 0;
@Override
public void run() {
counter++;
System.out.println("Running B1 -> " + counter);
}
}

public class ABMain {
public static void main(String[] args) throws InterruptedException {
A1 a = new A1(); // Create runnable instance
B1 b = new B1(); // Create Thread instance

new Thread(a).start();
```

```
Thread.sleep(1000);
new Thread(a).start();
Thread.sleep(1000);
new Thread(a).start();
Thread.sleep(1000);

new B1().start();
Thread.sleep(1000);
new B1().start();
Thread.sleep(1000);
new B1().start();
}
}

Output:

1
2
3


1
1
1
```

1) A1 a = new A1(); does not make a thread. It's just another class with no extra behavior. If you call A1.run() then it is not new thread. And A1.start() is not available to this class.
2) Only way to create start a thread in java is calling it's start method.
3) If you look at constructor of Thread class, no constructor takes parameter

of Thread class itself. And we know that Thread implements Runnable, effectively any call such as “new Thread(a).start();” is starting a thread with runnable mechanism.
4) Correct way to start thread (“using extend”) is calling it’s start method only directly.
e.g. new B1().start();
5) So effectively, in both techniques in your code, you are doing the same thing i.e. implementing runnable.

REPLY

satish Kumar Singh

APRIL 3, 2014 AT 11:44 AM

It’s correct post.

REPLY

Dhruv

NOVEMBER 27, 2014 AT 8:39 AM

By extending Thread, each of your threads has a unique object associated with it, whereas implementing Runnable, many threads can share the same runnable instance.

The statement is alright but what is the advantage we gain here ?

REPLY

Sanchita Vijayvargiya

NOVEMBER 13, 2013 AT 4:03 AM

Thank you for the post. It’s very helpful.

REPLY

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

nine + 9 =

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

☐ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,666 other subscribers

Email Address

Email Address

SUBSCRIBE

REFERENCES

Oracle Java Docs

Spring 3 Reference

Spring 4 References

Jboss RESTEasy JAX-RS

Hibernate Developer Guide

Junit Wiki

Maven FAQs

Dzone Dev Links

JSP Homepage

ANT Homepage

META LINKS

Advertise

Share your knowledge

Privacy policy

Contact Us

About Us

POPULAR POSTS

Spring 3 and hibernate integration tutorial with example

Reading/writing excel files in java : POI tutorial

How hashmap works in java

Singleton design pattern in java

Useful java collection interview questions

Understanding hibernate first level cache with example

How hibernate second level cache works?

Working with hashCode and equals methods in java

How to make a java class immutable

4 ways to schedule tasks in Spring 3 : @Scheduled example