Second method to achieve Multi-threading

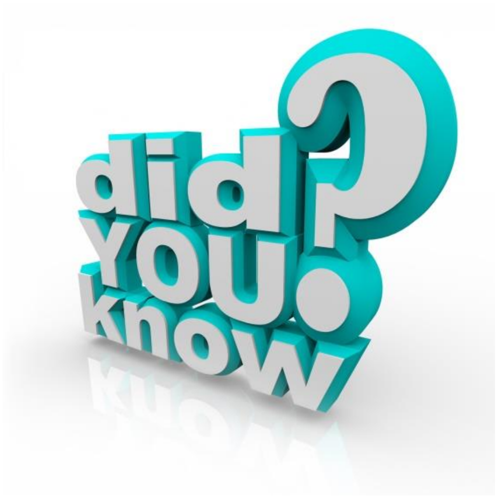
- ### Implementing a runnable Interface

Java runnable is an interface used to **execute code on a concurrent thread**. It is an interface which is implemented by any class if we want that the instances of that class should be executed by a thread.

The runnable interface has an undefined method **run()** with void as return type, and it takes in no arguments. The method summary of the run() method is given below-

Method	Description
public void run()	This method takes in no arguments. When the object of a class implementing Runnable class is used to create a thread, then the run method is invoked in the thread which executes separately.

The runnable interface provides a standard set of rules for the instances of classes which wish to execute code when they are active. **The most common use case of the Runnable interface is when we want only to override the run method.** When a thread is started by the object of any class which is implementing Runnable, then it invokes the run method in the separately executing thread.



A class that implements Runnable runs on a different thread without subclassing Thread as it instantiates a Thread instance and passes itself in as the target. This becomes important as classes should not be subclassed unless there is an intention of modifying or enhancing the fundamental behavior of the class.

Runnable class is extensively used in network programming as each thread represents a separate flow of control. **Also in multi-threaded programming, Runnable class is used.** This interface is present in **java.lang** package.

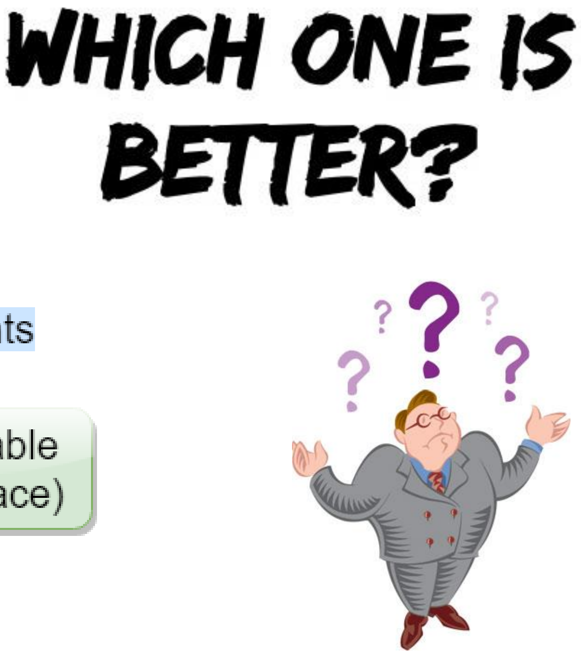
Implementing Runnable

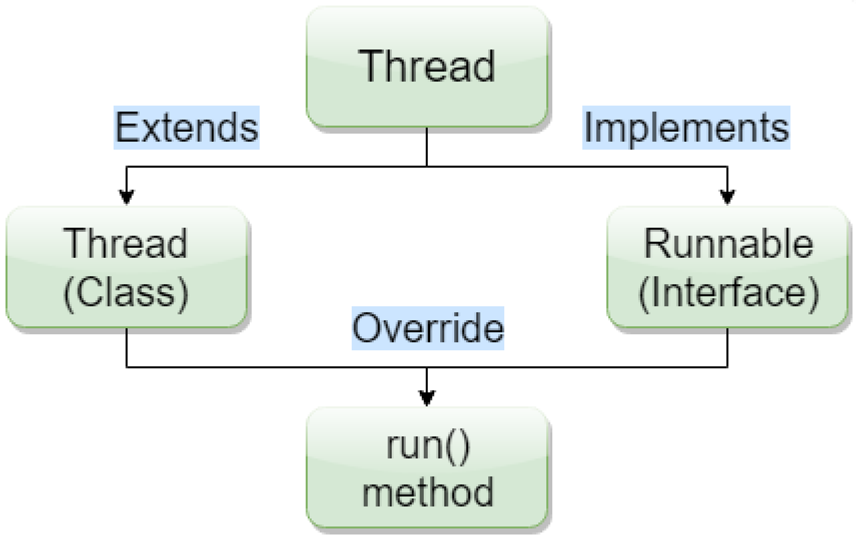
It is the easiest way to create a thread by implementing Runnable. One can create a thread on any object by implementing Runnable. To implement a Runnable, one has only to implement the run method.

public void run()

In this method, we have the code which we want to execute on a concurrent thread. In this method, we can use variables, instantiate classes, and perform an action like the same way the main thread does. The thread remains until the return of this method. The run method establishes an entry point to a new thread.

Thread vs. Runnable





There are several differences between Thread class and Runnable interface based on their performance, memory usage, and composition.

- By **extending thread, there is overhead of additional methods**, i.e. they consume excess or indirect memory, computation time, or other resources.

- Since in Java, we can only extend one class, and therefore **if we extend Thread class, then we will not be able to extend any other class**. That is why we should **implement Runnable interface to create a thread**.
- **Runnable makes the code more flexible**, if we are extending a thread, then our code will only be in a thread whereas, **in case of runnable, one can pass it in various executor services, or pass it to the single-threaded environment.**
- Maintenance of the code is easy if we implement the Runnable interface.
- We can **achieve basic functionality of a thread** by extending Thread class because it **provides some inbuilt methods like yield(), interrupt()** etc. that are not available in Runnable interface.

We can clearly see from the above points that implementing runnable is better approach.

Multi-threading using single run()

Previously we saw how to achieve multi-threading with multiple run() or by overriding run(). Now let's see how to achieve it by making use of single run().

Let us try to understand using the same example,

```
import java.util.Scanner;
class Demo1 implements Runnable
{
    public void run()
    {
        Thread t = Thread.currentThread();
        String name = t.getName();
        if(name.equals("ADD")==true)
        {
            add();
        }
        else if(name.equals("CHAR")==true)
        {
            charPrint();
        }
        else
        {
            numPrint();
        }
    }
```

```
    void add()
    {
        System.out.println("Addition task started");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        int c = a+b;
        System.out.println(c);
        System.out.println("Addition task completed");
    }

    void charPrint()
    {
        System.out.println("Character printing task");
        for(int i=65;i<=75;i++)
        {
            System.out.println((char)i);
            try
            {
                Thread.sleep(4000);
            }
            catch (Exception e)
            {
                System.out.println("Some problem occured");
            }
        }
        System.out.println("Character printing task completed");
    }


    void numPrint()
    {
        System.out.println("Number printing task started");
        for(int i=1;i<=10;i++)
        {
            System.out.println(i);
            try
            {
                Thread.sleep(4000);
            }
            catch (Exception e)
            {
                System.out.println("Some problem occured");
            }
        }
        System.out.println("Number printing task completed");
    }
}
```

```java
class Demo
{
    public static void main(String[] args)
    {
        Demo1 d1 = new Demo1();

        Thread t1 = new Thread(d1);
        Thread t2 = new Thread(d1);
        Thread t3 = new Thread(d1);

        t1.setName("ADD");
        t2.setName("CHAR");
        t3.setName("NUM");

        t1.start();
        t2.start();
        t3.start();
    }
}
```