Thread and runnable interfaces

Object Oriented Programming
2022 First Semester
Shin-chi Tadaki (Saga University)

Today’s theam

- Thread and runnable interfaces
- Synchronization between threads
- Protection by ”synchronized” keyword

Sample program download

`https://github.com/oop-mc-saga/Thread`

Threads

- Threads are mechanism to divide an application into separated processes executable asynchronously
- Threads can share the same variables
- In java applications
  - GUI class instances are running on threads
  - Any class instances can be executed on threads

Runnable interface

- Classes with the `Runnable` interface can be executed on threads
- `Runnable` interface has only one method `run()`, called only once from a thread
- Controlling variables for `run()` should be *volatile*
  - *Volatile* variables can be updated immediately

Methods of Thread class

- `start()`
    - Execute `run()` method of a specified instance
- `sleep()`
    - Sleep the thread during the specified time (millisecond)
- `stop()` method is obsolete and should not be used.

Two methods for defining a class runnable on thread

- By implementing the `Runnable` interface
- Define an anonymous class extending `Runnable`.
- Both methods need to implement `run()` method

Example of `Runnable` implementation

- `SampleWithThread`
  - Start the instance as an implementation of the `Runnable` interface
- `SampleRunnable`
  - Implement the `Runnable` interface

See `Thread.example0`

Sample class

```java
public class Sample {

    protected volatile boolean running = true;
    protected int c = 0;
    private final int id;

    public Sample(int id) {
        this.id = id;}

    public void update() {
        Date date = new Date();
        System.out.println(id + ":" + c + " "
            + date.toString());
        c++;
        if (c > 10) {
            running = false;
        }
    }

    public boolean isRunning() {
        return running;}
}
```

SampleWithThread class

```java
public static void main(String[] args) {
    Thread thread0 = new Thread(new Runnable() {
        Sample s = new Sample(1);

        public void run() {
            while (s.isRunning()) {
                s.update();
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                }
            }
        }
    });
    thread0.start();
}
```

This example defines an anonymous instance of `Runnable` class. Inside the definition, an instance of `Sample` class is created and `run()` method is defined.

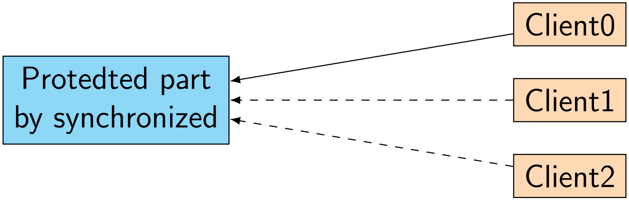
SampleRunnable class

```java
public class SampleRunnable extends Sample implements Runnable {

    public SampleRunnable(int id) {
        super(id);
    }

    /**
     * update() at random timing
     */
    @Override
    public void run() {
        while (running) {
            update();
            int t = (int) (1000 * Math.random());
            try {
                Thread.sleep(t);
            } catch (InterruptedException e) {
            }
        }
    }
    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        new Thread(new SampleRunnable(1)).start();
        new Thread(new SampleRunnable(2)).start();
        Thread t = new Thread(new SampleRunnable(3));
        t.start();
    }

}
```

Synchronization: 同期

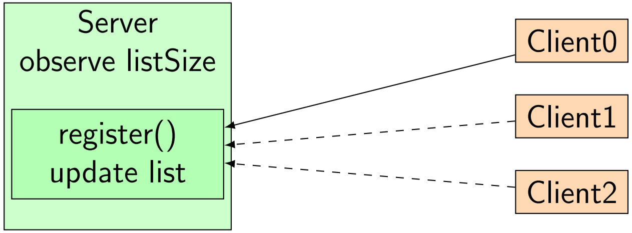
- Threads are allowed to update shared data in an application.
  - Applications need to synchronize updates of shared data as necessary.
- How to protect methods and objects
  - `synchronized` modifier
  - Only one thread is allowed to access the method/object.

Protection with `synchronized`



Only one of clients is allowed to access the resource.

Thread.example1



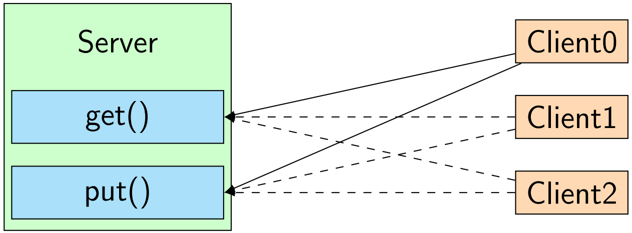
- Clients try to connect `register()` method by random duration.
- Only of of the clients is allowed to connect.

See `Thread.example1`

```java
public void run() {
    while (running) {
        //waiting the list unlocked
        synchronized (messageList) {
            if (messageList.size() == max) {
                running = false;
            }
        }
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
        }
    }
}
```

```java
synchronized public void register(Client client,
        int c, String dateStr) {
    Date date = new Date();
    //The time the client tries to connect and succeeds to connect
    String ss = client + ":" + c + " "
            + dateStr + "->" + date.toString();
    messageList.add(ss);
    System.out.println(ss);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
    }
}
```

Thread.example2



- The number of tokens equals to the number of clients.
- Clients try to get a token through `get()` method by random duration.
- After returning the token through `put()` method, the client is allowed to get another token.

See `Thread.example2`

Client side

```java
private void update(){
    if(!tokens.isEmpty()){//put token if this has
        running=server.put(this, tokens.poll());
    }
    Token t = server.get(this);//get token from the server
    if(t!=null){
        if(t==Server.falseToken)running=false;
        else{
            tokens.add(t);
        }
    }
}
```

Server side

```java
synchronized public Token get(Client client) {
    Token b = getSub(client);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
    }
    return b;
}
```

```java
synchronized boolean put(Client client, Token t) {
    if (running) {
        putSub(client, t);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
    }
    return running;
}
```

Exercise

In `example0/SampleRunnable.java`, understand

- How to start the thread,
- When the thread stops,
- Which variable and method triggers the stop event.