|  |  |  |
| --- | --- | --- |
| **2019\_2\_java2\_13** | **학번 : 20165164** | **이름 : 지현한** |

* **내용 점검**

1. Thread 클래스 상속

|  |
| --- |
| class MultithreadExample1 {  public static void main(String args[]) {  Thread thread = new DigitThread(); // 스레드 생성  thread.start(); // 스레드를 시작  for (char ch = 'A'; ch <= 'Z'; ch++)  System.out.print(ch);  }  }  class DigitThread extends Thread {  public void run() {  for (int cnt = 0; cnt < 10; cnt++)  System.out.print(cnt); }} |
| **[실행 결과]** |

1. Thread 클래스 상속 – 람다식 사용

|  |
| --- |
| public class MultithreadExample1 {  public static void main(String args[]) {  Thread thread = new Thread( **() -> {**  **for (int cnt = 0; cnt < 10; cnt++)**  **System.out.print(cnt); //람다식 사용**  **}**);  thread.start(); // 스레드를 시작  for (char ch = 'A'; ch <= 'Z'; ch++)  System.out.print(ch);  }  } |
| **[실행결과]** |

1. Runnable 인터페이스 구현 – 익명 객체

|  |
| --- |
| public class TestThread {  public static void main(String[] args) {  Thread t1 = new Thread( **new Runnable() {**  **public void run() {**  **for (int cnt = 0; cnt < 10; cnt++)**  **System.out.print(cnt);**  **}**  **});**    t1.start();  for (char ch = 'A'; ch <= 'Z'; ch++)  System.out.print(ch);  }  } |
| **[실행결과]** |

1. Runnable 인터페이스 구현 – 람다식 활용

|  |
| --- |
| **public** **class** TestThread {  **public** **static** **void** main(String[] args) {  Runnable t1 = () -> {  **for** (**int** cnt = 0; cnt < 10; cnt++) {  System.***out***.print( "lamda"+ cnt);  **try** {  Thread.*sleep*(500); //InterruptedException 예외처리가 반드시 있어야 함  }  **catch**(InterruptedException e) {}  }  };  **new** Thread(t1).start();  **for** (**char** ch = 'A'; ch <= 'Z'; ch++)  System.***out***.print(ch);  }  } |
| **[실행결과]** |

1. 임계 영역과 동기화 : “Lecture3\_자바API\_예외처리\_스레드” 수업자료 슬라이드 82~84

|  |
| --- |
| **[실행결과]** |

* **내용 점검**

1. ArrayIndexOutOfBoundsException 예외를 발생시키는 프로그램을 작성하고 try/catch 블록을 이용하여 처리하여 보라.

double[] d=new double[10];

try {

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

catch (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) {

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; // 콘솔에 오류 메시지를 출력한다.

}

**정답] double**[] d=**new** **double**[10];

**try** {

d[10]=1.0;

}**catch**(ArrayIndexOutOfBoundsException e) {

System.***out***.println("배열 인덱스를 벗어났습니다.");

}

1. 다음과 같은 예외 처리기로 잡을 수 있는 예외는 어떤 종류인가? 그리고 이런 종류의 예외 처리기가 바람직하지 않은 이유는?

catch (Exception e) {

...

}

**정답**. 모든 예외를 처리한다, 어떤 예외가 발생 했는지 알 수 없다.

1. 다음 코드에서 잘못된 부분을 지적하시오.

try {

...

} catch (Exception e) {

...

} catch (ArithmeticException a) {

...

}

**정답]** 범위가 작은 예외를 먼저 처리해야한다**.**

1. 다음 프로그램의 출력을 쓰시오.

try {

int n = Integer.parseInt("abc");

System.out.println("try");

}

catch (NumberFormatException e){

System.out.println("숫자 형식 오류");

}

finally {

System.out.println("finally");

}

**정답]**



1. 다음 프로그램의 출력을 쓰시오.

try {

int[] array=new int[-10];

System.out.println("try");

}

catch (NumberFormatException e){

System.out.println("숫자 형식 오류");

}

catch (NegativeArraySizeException e){

System.out.println("배열 크기 음수 오류");

}

catch (Exception e){

System.out.println("오류");

}

**정답]** 

1. 다음 프로그램의 출력을 쓰시오.

public class Test{

public static void throwit (){

System.out.print("A ");

throw new RuntimeException();

}

public static void main(String [] args){

try{

System.out.print("B ");

throwit();

}

catch (Exception re ){

System.out.print("C ");

}

finally{

System.out.print("D ");

}

System.out.println("E ");

}

}

**정답]** 

1. 다음 프로그램을 컴파일이 되도록 올바르게 수정하시오.

try {

int x = 0;

int y = 5 / x;

}

catch (Exception e){

System.out.println("Exception");

}

catch (ArithmeticException ae){

System.out.println(" Arithmetic Exception");

}

System.out.println("finished");

**정답]**

**try** {

**int** x = 0;

**int** y = 5 / x;

} **catch** (ArithmeticException ae) {

System.***out***.println(" Arithmetic Exception");

} **catch** (Exception e) {

System.***out***.println("Exception");

}

System.***out***.println("finished");

1. 다음 프로그램의 출력은?

public class Test{

public static void main(String args[]){

try{

System.out.print("Hello world ");

}

finally{

System.out.println("Finally executing ");

}

}

}

**정답]** 

1. 다음 프로그램을 컴파일하여 보자.

public class Test {

public static void main(String[] args) {

sub();

}

public static void sub() {

int[] array = new int[10];

int i = array[10];

}

}

(1) 위의 프로그램은 컴파일 시에 오류가 발생한다. 어떤 오류가 발생하는가?

ArrayIndexOutOfBoundsException

(2) try/catch 블록을 사용하여서 예외를 처리하여 보라.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**try** {

*sub*();

} **catch** (Exception e) {

System.***out***.println("인덱스 오류");

}

}

**public** **static** **void** sub() {

**int**[] array = **new** **int**[10];

**int** i = array[10];

}

}

(3) throws 선언을 이용하여 예외를 처리하여 보라.

**정답]**

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**try** {

*sub*();

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

**public** **static** **void** sub() **throws** Exception {

**int**[] array = **new** **int**[10];

**int** n = 10;

**if** (n >= array.length) {

**throw** **new** Exception("인덱스 오류");

}

**int** i = array[n];

}

}

1. 다음은 Stack 클래스의 일부분이다.

(1) Stack에 저장되는 데이터의 타입을 int 대신에 제네릭 타입으로 표시하여 보자.

public class Stack{

private int[] stack;

public void push(int data) { ....}

public int pop() { ....}

}

(2) String 타입의 데이터를 가지는 Stack을 생성하는 문장을 쓰시오.

**정답 1~2번]**

**public** **class** Stack<T> {

**private** T[] stack;

**public** **void** push(T data) {}

**public** **int** pop() {}

}

Stack<String> stack = **new** Stack<String>();

1. 스레드의 몸체를 포함하는 메소드는 무엇인가?
2. run();
3. start();
4. stop();
5. main();

**정답]**

run()

1. 다음은 스레드를 생성하여 실행하는 코드이다. 비어 있는 부분에 어떤 코드를 넣어야 할까?

class Test implements Runnable {

public static void main(String args[]) {

/\* 비어있는 부분 \*/

}

public void run() {}

}

**정답]**

Thread t = **new** Thread(**new** Test());

t.start();

1. 다음의 메소드는 어떠한 경우에 사용되는가?

(1) sleep() : 일정주기로 스레드를 실행할 때

(2) yield() : CPU를 다른 스레드에게 양보

(3) start() : 스레드 시작

1. 다음 프로그램의 출력은?

public class Test implements Runnable{

public static void main(String [] args){

Test t = new Test();

Thread x = new Thread(t);

x.start();

}

public void run(){

for(int i = 0; i < 3; ++i){

System.out.print(i + "..");

}

}

}

**정답]** 

1. 다음 프로그램의 출력은?

public class Test {

private int count = 1;

public synchronized void sub() {

for (int i = 0; i < 10; i++)

System.out.println(count++);

}

public static void main(String[] args) {

Test demo = new Test();

Thread a1 = new A(demo);

Thread a2 = new A(demo);

a1.start();

a2.start();

}

}

class A extends Thread {

Test demo;

public A(Test td) {

demo = td;

}

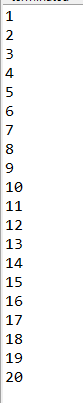
public void run() {

demo.sub();

}

}

**정답]**



1. 질문에 답하시오

(1) 출력을 예상하여 보라.

**class** Job **implements** Runnable {

**public** **void** run() {

**byte** n = 0;

**while**(**true**)

System.*out*.println(""+n++);

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** Job());

t.start();

}

}

(2) Thread 클래스를 상속받는 형태로 다시 작성하라.

(3) Job 클래스의 생성자를 작성하고 매개 변수로 스레드의 이름을 문자열 형태로 받도록 하라. 스레드가 실행되면서 스레드의 이름을 출력한다.

(4) 동일한 스레드를 하나 더 생성하여 동시에 실행하여 보자. 어떤 출력이 나타나는가? 출력을 구분하기 위하여 스레드의 이름을 다르게 하라.

**정답 1~4 번]**

1. -2^7 ~ 2^7-1 의 수가 무한이 출력된다
2. **class** Job **extends** Thread {

**public** **void** run() {

**byte** n = 0;

**while** (**true**)

System.***out***.println("" + n++);

}

}

**public** **class** Test5 {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** Job());

t.start();

}

}

1. **class** Job **extends** Thread {

String name;

Job(String name) {

**this**.name = name;

}

**public** **void** run() {

System.***out***.println(name);

}

}

**public** **class** Test5 {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** Job("t"));

t.start();

}

}

1. **class** Job **extends** Thread {

String name;

Job(String name) {

**this**.name = name;

}

**public** **void** run() {

System.***out***.println(name);

}

}

**public** **class** Test5 {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** Job("t"));

Thread h = **new** Thread(**new** Job("h"));

t.start();

h.start();

}

}

1. 다음 프로그램의 출력을 예상하여 보라.

**class** MyJob **implements** Runnable {

**public** **void** run() {

**try** {

System.*out*.println("A");

Thread.*sleep*(1000);

System.*out*.println("B");

} **catch** (InterruptedException e) {

System.*out*.println("C");

}

System.*out*.println("D");

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** MyJob());

t.start();

t.interrupt();

}

}

**정답]**



* **실습 과제**

1. 다음과 같이 특수 문자와 영문자를 출력하는 스레드를 작성하고 테스트 하는 프로그램을 작성하세요. 단, 출력은 0.5초 후에 수행되도록 한다. (힌트: sleep() 메소드 사용)

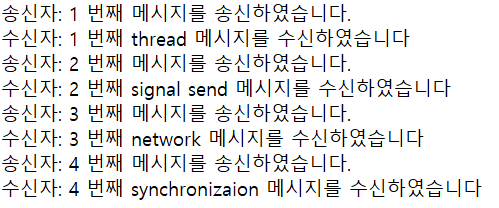
|  |  |
| --- | --- |
|  | **public** **class** TestThread {  **public** **static** **void** main(String[] args) {  Thread th1=**new** Thread(**new** Thread1());  Thread th2=**new** Thread(**new** Thread2());  th1.start();  th2.start();  }  } |

|  |
| --- |
| **[프로그램 소스]**  **class** Thread1 **implements** Runnable {  **char**[] ch = { '◆', '▣', '※', '○', '♠' };  **public** **void** run() {  **for** (**char** temp : ch)  System.***out***.println("lamda : " + temp);  }  }  **class** Thread2 **implements** Runnable {  **public** **void** run() {  **char** ch = 'b';  **while** (ch + 2 <= 't') {  System.***out***.println("lamda : " + (ch += 2));  }  }  } |
| **[실행결과]** |

1. 위 1)번 문제를 람다식을 사용하여 수정한 후 결과를 제시하시오.

|  |
| --- |
| **[프로그램 소스]**  **public** **class** TestThread {  **public** **static** **void** main(String[] args) {  Thread th1 = **new** Thread(()-> {  **char**[] ch = { '◆', '▣', '※', '○', '♠' };  **for** (**char** temp : ch)  System.***out***.println("lamda : " + temp);  });  Thread th2 = **new** Thread(**new** Thread(()->{  **char** ch = 'b';  **while** (ch + 2 <= 't') {  System.***out***.println("lamda : " + (ch += 2));  }  }  ));  th1.start();  th2.start();  }  } |
| **[실행 결과]** |

1. 내용 점검 5)번 예제 프로그램을 다음과 같이 실행될 수 있도록 수정하시오. 단, 전송할 메시지는 다음과 같다 **new** String[] {"thread", "signal send","network", "synchronizaion"}



|  |
| --- |
| **[프로그램 소스]**  **class** Buffer {  **private** String data; // 생산자로부터 소비자로 전해지는 데이터  **private** **boolean** empty = **true**; // 소비자가 기다리면 true, 생산자가 기다리면 false  **public** **synchronized** String get() {  **while** (empty) {  **try** {  wait(); // 리턴 한 후 반드시 조건을 다시 검사 해야 함  } **catch** (InterruptedException e) {  }  }  empty = **true**; // 상태를 토글한다  notifyAll(); // 생산자를 깨운다  **return** data;  }  **public** **synchronized** **void** put(String data) {  **while** (!empty) {  **try** {  wait();  } **catch** (InterruptedException e) {  }  }  empty = **false**;  **this**.data = data;  notifyAll();  }  }  **class** send **implements** Runnable {  **private** Buffer buffer;  String[] str = { "thread", "singnal send", "network", "synchronizaion" };  **public** send(Buffer buffer) {  **this**.buffer = buffer;  }  **public** **void** run() {  **for** (**int** i = 0; i < str.length; i++) {  buffer.put(str[i]);  System.***out***.printf("송신자: %d 번째 메시지를 송신하였습니다.\n", i + 1);  **try** {  Thread.*sleep*((**int**) (Math.*random*() \* 100));  } **catch** (InterruptedException e) {  }  }  }  }  **class** recive **implements** Runnable {  **private** Buffer buffer;  String[] str = { "thread", "singnal send", "network", "synchronizaion" };  **public** recive(Buffer buffer) {  **this**.buffer = buffer;  }  **public** **void** run() {  **for** (**int** i = 0; i < str.length; i++) {  String data = buffer.get();  System.***out***.printf("수신자: %d 번째 %s 메시지를 수신하였습니다.\n", i + 1, str[i]);  **try** {  Thread.*sleep*((**int**) (Math.*random*() \* 100));  } **catch** (InterruptedException e) {  }  }  }  }  **public** **class** Q3main {  **public** **static** **void** main(String[] args) {  Buffer buffer = **new** Buffer();  (**new** Thread(**new** send(buffer))).start(); // 생산자 스레드 시작  (**new** Thread(**new** recive(buffer))).start(); // 소비자 스레드 시작  }  } |
| **[실행결과]** |

1. 생성된 난수 만큼 문자와 숫자를 번갈아 출력하는 스레드 프로그램을 작성하시오.





|  |
| --- |
| **[프로그램 소스]**  **public** **class** Q5main {  **public** **static** **void** main(String[] args) {  Q5Buffer buffer = **new** Q5Buffer();  (**new** Thread(**new** ch(buffer))).start();  (**new** Thread(**new** number(buffer))).start();  }  }  **class** Q5Buffer {  **private** **boolean** empty = **true**;  **public** **synchronized** **void** get() {  **while** (empty) {  **try** {  wait();  } **catch** (InterruptedException e) {  }  }  empty = **true**;  notifyAll();  }  **public** **synchronized** **void** put() {  **while** (!empty) {  **try** {  wait();  } **catch** (InterruptedException e) {  }  }  empty = **false**;  notifyAll();  }  }  **class** ch **implements** Runnable {  **private** Q5Buffer buffer;  String str = "◇ □";  **public** ch(Q5Buffer buffer) {  **this**.buffer = buffer;  }  **public** **void** run() {  String print = "";  **for** (**int** i = 0; i < 2; i++) {  **for** (**int** j = 0; j < Math.*random*() \* 10; j++) {  print += str;  }  System.***out***.print(print);  buffer.put();  **try** {  Thread.*sleep*((**int**) (Math.*random*() \* 100));  } **catch** (InterruptedException e) {  }  }  }  }  **class** number **implements** Runnable {  **private** Q5Buffer buffer;  String num = String.*valueOf*((**int**) (Math.*random*() \* 10));  **public** number(Q5Buffer buffer) {  **this**.buffer = buffer;  }  **public** **void** run() {  String print = "";  **for** (**int** i = 0; i < 2; i++) {  **for** (**int** j = 0; j < Math.*random*() \* 10; j++) {  print += num;  }  System.***out***.print(print);  buffer.get();  **try** {  Thread.*sleep*((**int**) (Math.*random*() \* 100));  } **catch** (InterruptedException e) {  }  }  }  } |
| **[실행 결과]** |