Exercise 2:

a.2

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
child Thread
```

```
b.2

c.2
No arg method

d. 2,No arg method
Main method

e.2, start method
No arg method
Main method

f.2, Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread

case1:
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

case 2:

```
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

case3:

```
main thread
main thread
main thread
main thread
main thread
main thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
main thread
main thread
main thread
main thread
```

case 4:

```
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

case 5:
error variable r of type MyRunnable

case 6:

```
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
Child thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
main thread
```

q1.3,4

q2.3,4

q3.3

analyze:

```
1.main
Thread-0
Bhaskar Thread
```

2.
No comment:
```
main thread
main thread
main thread
main thread
main thread
child thread
child thread
child thread
child thread
child thread
```

comment:
```
main thread
main thread
main thread
main thread
main thread
child thread
child thread
child thread
child thread
child thread
```

3.
Uncomment:
```
Sita Thread
Sita Thread
Sita Thread
Sita Thread
Sita Thread
Rama Thread
Rama Thread
Rama Thread
Rama Thread
Rama Thread
```

Comment:
```
Rama Thread
Rama Thread
Rama Thread
Rama Thread
Rama Thread
Sita Thread
Sita Thread
Sita Thread
Sita Thread
Sita Thread
```

4.
Uncmt:

```
end of main thread
i am lazy Thread :0
i got interrupted
```

cmmt:

```
end of main thread
i am lazy Thread :0
i am lazy Thread :1
i am lazy Thread :2
i am lazy Thread :3
i am lazy Thread :4
i am lazy Thread :5
i am lazy Thread :6
i am lazy Thread :7
```

5. end of main thread

```
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
iam lazy thread
I'm entered into sleeping stage
i got interrupted
```

Find output:

```
Thread1 starts execution of foo()method
Thread2 starts execution of bar() method
```

```
Thread1 trying to call b.last()
Thread2 trying to call a.last()
```

```
6. good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:dhoni
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj
good morning:yuvaraj
```

Timeout

Collections :
Array:
search an element in a array list.

```
import java.util.ArrayList;

public class SearchElementInArrayList {
    public static void main(String[] args) {
        ArrayList<Integer> arrayList = new ArrayList<>();
        arrayList.add(1);
        arrayList.add(2);
        arrayList.add(3);
        arrayList.add(4);
        int elementToSearch = 3;
        boolean found = arrayList.contains(elementToSearch);
        System.out.println("Element " + elementToSearch + " found: " + found);
    }
}
```

swap two elements in an ArrayList:

```
import java.util.ArrayList;
import java.util.Collections;

public class SwapElements {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);

        int index1 = 1;
        int index2 = 3;

        Collections.swap(list, index1, index2);

        System.out.println("ArrayList after swapping: " + list);
    }
}
```

Join two arraylist:

```
import java.util.ArrayList;

public class JoinArrayLists {
    public static void main(String[] args) {
        ArrayList<String> list1 = new ArrayList<>();
        list1.add("A");
        list1.add("B");

        ArrayList<String> list2 = new ArrayList<>();
        list2.add("C");
        list2.add("D");

        list1.addAll(list2);

        System.out.println("Joined ArrayList: " + list1);
    }
}
```

Java program to replace the second element of a ArrayList with the specified element:

```
import java.util.ArrayList;

public class ReplaceSecondElement {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Aman");
        list.add("Hello");
        list.add("bye");

        String replacement = "Hello";
        list.set(1, replacement);

        System.out.println("ArrayList after replacement: " + list);
    }
}
```

LinkedList Complete:

```
import java.util.LinkedList;

public class LinkedListOps {
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<>();

        append(list, "Aman");
        append(list, "Hello");
        append(list, "phone");

        System.out.println("Linked List after appending: " + list);

        pop(list);

        System.out.println("Linked List after popping: " + list);

        update(list, "Hello", "Bye");

        System.out.println("Linked List after updating: " + list);

        find(list, "phone");
    }

    private static void append(LinkedList<String> list, String element) {
        list.add(element);
    }

    private static void pop(LinkedList<String> list) {
        if (!list.isEmpty()) {
            list.removeLast();
        } else {
            System.out.println("Cannot pop from an empty list.");
        }
    }

    private static void update(LinkedList<String> list, String oldElement, String newElement) {
        if (list.contains(oldElement)) {
            int index = list.indexOf(oldElement);
            list.set(index, newElement);
        } else {
            System.out.println("Element not found in the list.");
        }
    }

    private static void find(LinkedList<String> list, String element) {
        if (list.contains(element)) {
            System.out.println(element + " found in the list.");
        } else {
            System.out.println(element + " not found in the list.");
        }
    }
}
```

HashSet Complete:

```
import java.util.HashSet;

public class HashSetOperations {
    public static void main(String[] args) {
        HashSet<String> hashSet = new HashSet<>();

        hashSet.add("Tab");
        hashSet.add("watch");
        hashSet.add("phone");
        String elementToFind = "watch";
        boolean found = hashSet.contains(elementToFind);
        System.out.println("Element " + elementToFind + " found: " + found);

        String elementToRemove = "watch";
        hashSet.remove(elementToRemove);
        String updatedElement = "laptop";
        hashSet.add(updatedElement);

        System.out.println("HashSet after update: " + hashSet);

        String elementToPop = "phone";
        hashSet.remove(elementToPop);

        System.out.println("HashSet after popping: " + hashSet);
    }
}
```

Stack:

```
import java.util.Stack;

public class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for (char bracket : s.toCharArray()) {
            if (bracket == '{' || bracket == '(' || bracket == '[') {
                stack.push(bracket);
            } else {
                if (stack.isEmpty()) {
                    return false;
                }
                char n = stack.pop();
                if (bracket == ')' && n != '(' ||
                    bracket == '}' && n != '{' ||
                    bracket == ']' && n != '[') {
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        System.out.println(solution.isValid("({[()]})"));
    }
}
```

2.

```
import java.util.HashSet;
import java.util.Set;
import java.util.Stack;

public class StackSubjects {
    public static void main(String[] args) {
        Stack<String> stack1 = new Stack<>();
        Stack<String> stack2 = new Stack<>();

        stack1.addAll(Set.of("Chemistry", "Mathematics", "Biology", "English"));
        stack2.addAll(Set.of("Biology", "English", "Geography", "Physics"));

        Set<String> onlyInStack1 = new HashSet<>(stack1);
        onlyInStack1.removeAll(stack2);

        Set<String> onlyInStack2 = new HashSet<>(stack2);
        onlyInStack2.removeAll(stack1);

        Set<String> commonSubjects = new HashSet<>(stack1);
        commonSubjects.retainAll(stack2);

        System.out.println("Only present in the first stack: " + onlyInStack1);
        System.out.println("Only present in the second stack: " + onlyInStack2);
        System.out.println("Present in both stacks: " + commonSubjects);
    }
}
```

TreeSet:

```
import java.util.TreeSet;
import java.util.Comparator;

public class TreeSetDescendingOrder {
    public static void main(String[] args) {
        TreeSet<Integer> treeSet = new TreeSet<>(Comparator.reverseOrder());

        treeSet.add(3);
        treeSet.add(2);
        treeSet.add(5);
        treeSet.add(1);

        System.out.println("Elements in descending order:");
        for (Integer num : treeSet) {
            System.out.println(num);
        }
    }

}
```

Queue:

a.

```
import java.util.LinkedList;
import java.util.Queue;

public class PrimeNumberQueue {
    public static void main(String[] args) {
        Queue<Integer> primeQueue = new LinkedList<>();

        for (int i = 2; i <= 20; i++) {
            if (isPrime(i)) {
                primeQueue.offer(i);
            }
        }

        System.out.println("Prime numbers in the queue:");
        while (!primeQueue.isEmpty()) {
            System.out.println(primeQueue.poll());
        }
    }

    private static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

b.

```
import java.util.PriorityQueue;

public class PriorityQueueExample {
    public static void main(String[] args) {
        // Ascending Order
        PriorityQueue<Integer> ascendingQueue = new PriorityQueue<>();
        ascendingQueue.offer(1);
        ascendingQueue.offer(3);
        ascendingQueue.offer(2);
        ascendingQueue.offer(4);
        System.out.println("Ascending Order Queue: " + ascendingQueue);

        // Descending Order
        PriorityQueue<Integer> descendingQueue = new PriorityQueue<>((a, b) -> b - a);
        descendingQueue.offer(2);
        descendingQueue.offer(4);
        descendingQueue.offer(5);
        descendingQueue.offer(1);
        System.out.println("Descending Order Queue: " + descendingQueue);
    }
}
```