سوال اول

الف)

Wrapper class: این کلاس ها راهی برای استفاده از primitive typeها به عنوان object هستند.
زبان

جاوا برای هر primitive type یک wrapper class دارد که با آن مطابقت دارد.

به عنوان مثال:

boolean <-> Boolean

int <-> Int

float <-> Float

Autoboxing and Unboxing: به تبدیل کردن یک نوع داده اولیه(primitive type) به object از wrapper class متناظر با آن، Autoboxing در جاوا گفته میشود.

به عنوان مثال:

Integer five=5

که جاوا به صورت اتوماتیک Integer Constructorرا صدا میزند.

به تبدیل کردن یک آبجکت از نوع کلاس Wrapper به نوع داده اولیهای (primitive type) که متناظر آن است، Unboxing گفته میشود.

به عنوان مثال:

int six=five + 1

که جاوا به صورت اتوماتیک intValue را برای five صدا میزند.

ب)

1) خیر زیرا ممکن است در یک برنامه تعداد زیادی آبجکت بسازیم که با نگه داشتن reference به آنها، هیچگاه از scope خارج نشده و در نتیجه فرآیند Garbage Collection روی آنها انجام نشود و حافظه پر شود.

(2

1. Memory is allocated from heap to hold all instance variables and

implementation-specific data of the object and its super classes. Implementation specific data includes pointers to class and method data.

2. The instance variables of the objects are initialized to their default values.

3. The constructor for the most derived class is invoked. The first thing a

constructor does is to call the constructor for its super classes. This process

continues until the constructor for java.lang.Object is called, as java.lang.Object is

the base class for all objects in java.

4. Before the body of the constructor is executed, all instance variable initializers

and initialization blocks are executed. Then the body of the constructor is

executed. Thus, the constructor for the base class completes first and constructor for the most derived class completes last.

(*3*

● Basic for loop:

```
for (int i = 0; i < myList.size(); i++) {
System.out.println(myList.get(i));
}
```

● Enhanced for loop: The enhanced for loop is a simple structure that allows us to visit every element of a list.

```
for (String entry : myList) {
System.out.println(entry);
}
```

● Iterator: An Iterator is a design pattern that offers us a standard interface to traverse a data structure without having to worry about the internal representation.

```
Iterator<String>
myListIterator = myList.iterator();
while(myListIterator.hasNext()) {
     System.out.println(myListIterator.next());
}
```

● ListIterator:

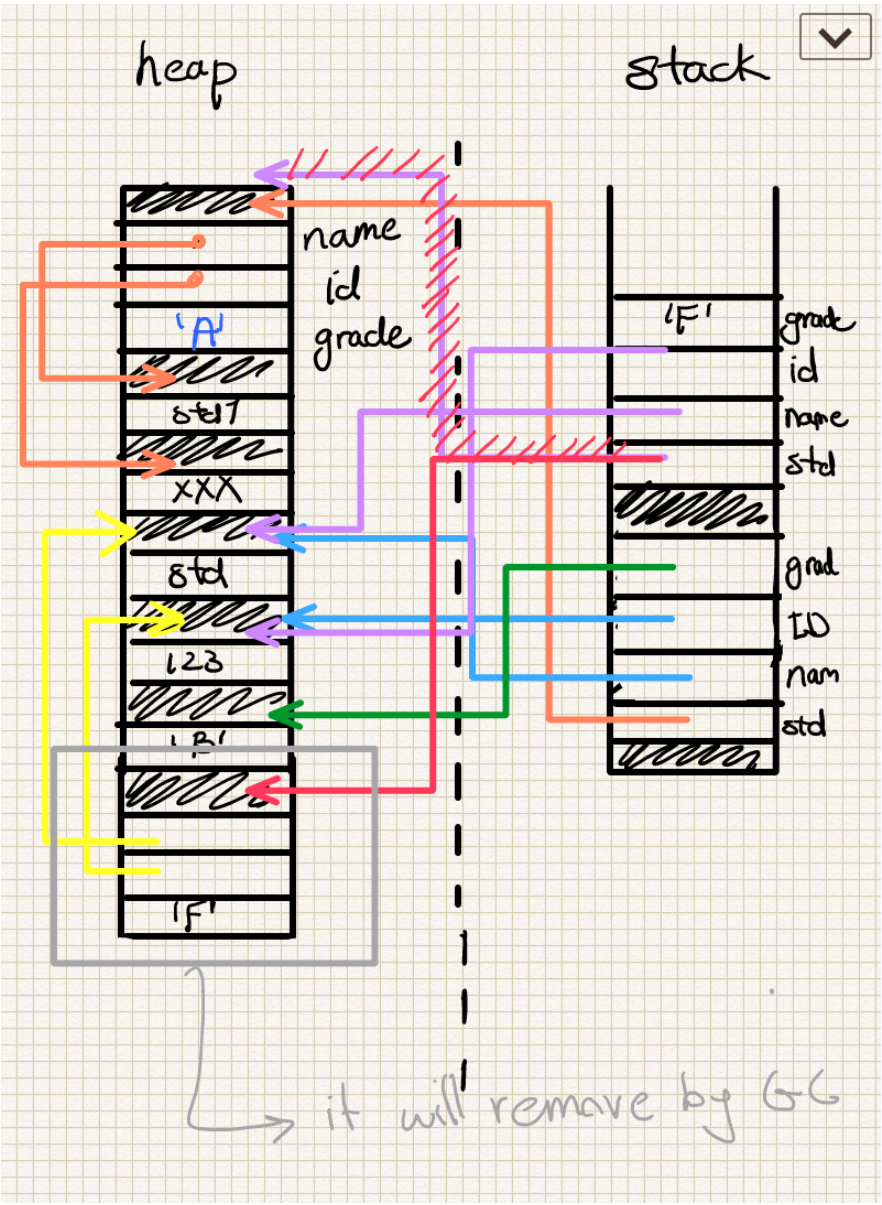
A ListIterator allows us to traverse a list of elements in either forward or backward order.

```
ListIterator<String>
listIterator= myList.listIterator();
while(listIterator.hasNext()) {
     System.out.println(listIterator.next());
}
```

اگر به جای مورد چهارم هر یک از موارد while, forEach.Iterable, forEach.Stream توضیح داده شده باشد قابل قبول است.

ج)

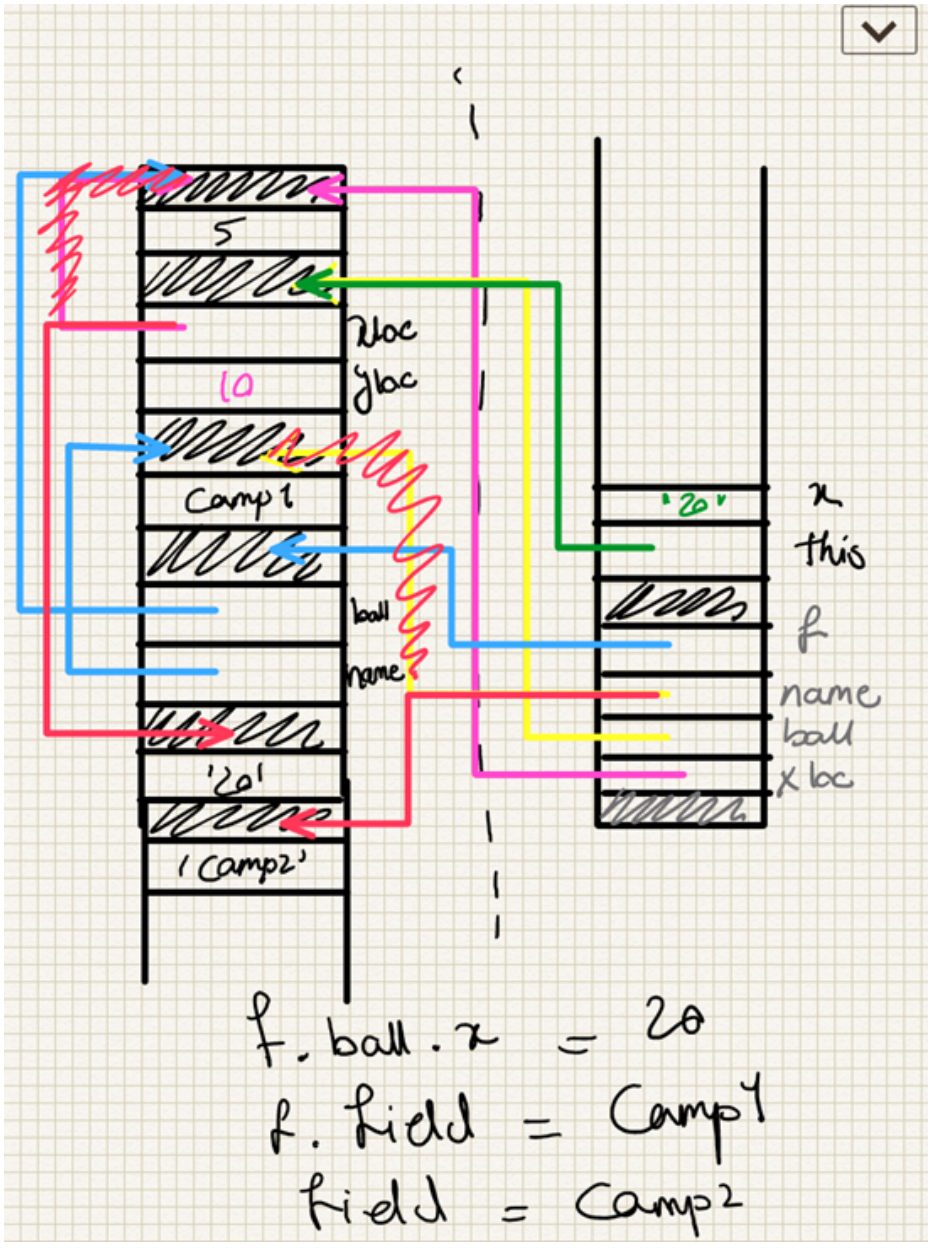
1. پاسخ: خیر


heap
stack
name
id
grade
'A'
std
XXX
std
123
'B'
'F'
'F'
grade
id
name
std
grad
ID
nam
std
it will remove by GC

f.ball.xLocation = 20

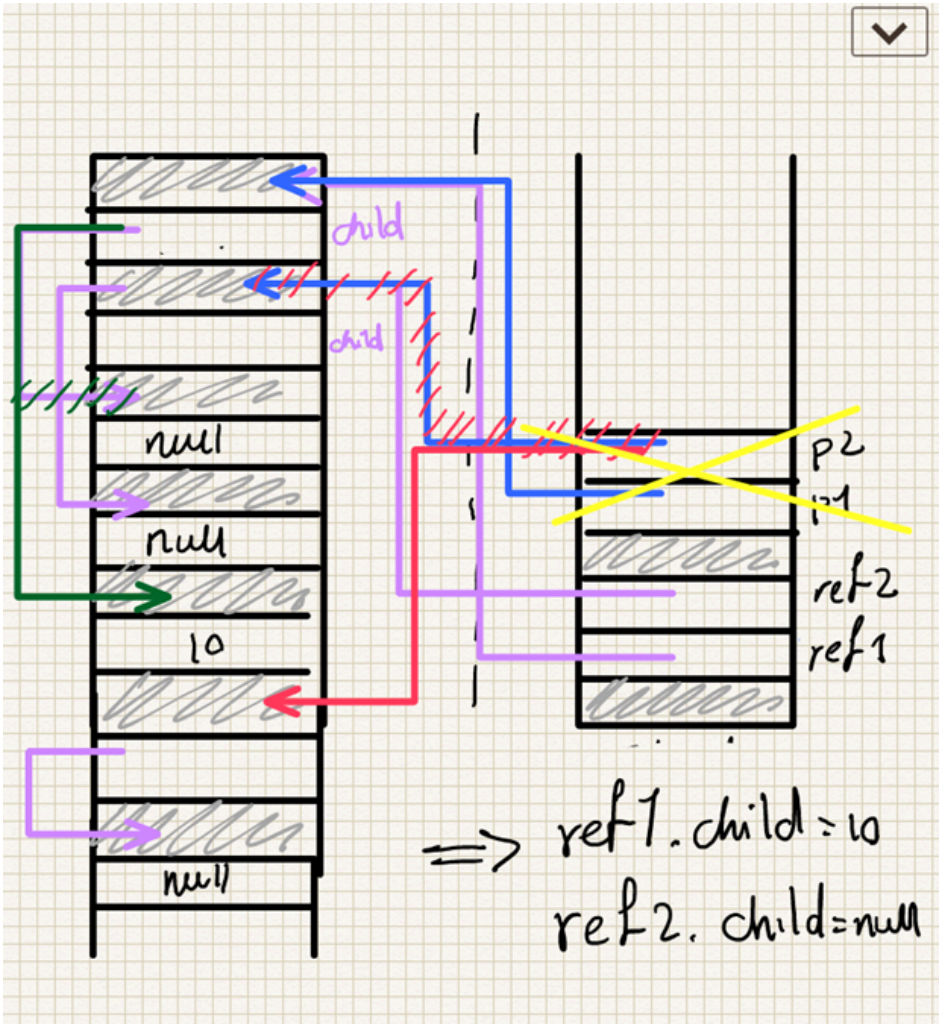
f.field = “Camp1”

Field = “Camp2”



ref1.child = 10

ref2.child = null



د)

Garbage Collection: در JVM یک Garbage Collector قرار دارد که وظیفه اش این است که فضاهایی در heap که هیچ پوینتری از stack به آنها اشاره نمی کند (unused object) را reclaim می کند تا دوباره از آنها در آینده استفاده کند. به این فرایند Garbage Collection گفته میشود.

```
public class Main {

	public static void main(String[] args) {

	Runtime r = Runtime.getRuntime();

	r.gc();



	long free1 = r.maxMemory() - r.totalMemory() + r.freeMemory();

	System.out.println("free memory before allocating: " + free1);



	for (int i = 0; i < 10000; i++) {

		new Rational(i + 1, i + 2);

	}



	long free2 = r.maxMemory() - r.totalMemory() + r.freeMemory();

	System.out.println("free memory after allocating: " + free2);

	System.out.println("memory used for 10000 Rational object: " + (free1 -
free2));



	r.gc();

	long free3 = r.maxMemory() - r.totalMemory() + r.freeMemory();

	System.out.println("free memory after calling GC: " + free3);

	System.out.println("memory freed by GC: " + (free3 - free2));
```

```
        }

}
```

خروجی:

```
free memory before allocating: 2117361024

free memory after allocating: 2116765272

memory used for 10000 Rational object: 595752

free memory after calling GC: 2117286408

memory freed by GC: 521136
```

سوال دوم

```java
import java.text.SimpleDateFormat;
import java.util.Date;

/**
* @author Parham Ahmady
* @since 14/3/2022
*/
public class Logger {

   private static final String INFO_SIGNATURE = "[INFO]";
   private static final String WARN_SIGNATURE = "[WARN]";
   private static final String ERROR_SIGNATURE = "[ERROR]";

   private final String className;

   /**
    * Private constructor to prevent instantiation from outside the class
    * @param className the name of the class that is using this logger
    */
   private Logger(String className) {
       this.className = className;
   }

   /**
    * This method is used to log a warning message
    * @param message the message to be logged
    * @param args the arguments to be replaced in the message
    */
   public void warn(String message, String... args) {
       System.out.println(buildLog(message, args, WARN_SIGNATURE));
   }

   /**
    * This method is used to log an info message
    * @param message the message to be logged
    * @param args the arguments to be replaced in the message
    */
   public void info(String message, String... args) {
       System.out.println(buildLog(message, args, INFO_SIGNATURE));
   }

   /**
```

```
     * This method is used to log an error message
     * @param message the message to be logged
     * @param args the arguments to be replaced in the message
     */
    public void error(String message, String... args) {
        System.out.println(buildLog(message, args, ERROR_SIGNATURE));
    }

    private String buildMessage(String message, String... args) {
        if (args == null || args.length == 0)
            return message;
        for (String arg : args) {
            int regexIndex = message.indexOf("{}");
            if (regexIndex < 0)
                return message;
            message=message.substring(0, regexIndex) + arg +
            message.substring(regexIndex + 2);
        }
        return message;
    }

    @SuppressWarnings("StringBufferReplaceableByString") //just to make it look
    better in intellij IDEA
    private String buildLog(String message, String[] args, String signature) {
        // stringBuilder is used to build the log message, there are many ways to
           do this
        StringBuilder builder = new StringBuilder(getCurrentDate())
                .append(" ")
                .append(signature)
                .append(" ")
                .append(className)
                .append(" \n\t")
                .append(buildMessage(message, args));
        return builder.toString();
    }

    private String getCurrentDate() {
        //this is an easy way to get the current date and time, there are many
          ways to do this
        return new SimpleDateFormat("yyyy-MM-dd' 'hh:mm:ss:SS")
                    .format(new Date());
    }

    /**
     * This method is used to create a new logger instance (Called FactoryMethod)
     * @param className the name of the class that is using this logger
     * @return the logger instance
```

```
     */
    public static Logger getLogger(String className) {
        if (className != null && !className.isEmpty())
            return new Logger(className);
        return null;
    }
}
```