16-1-24

LAB - 6

1. Demonstrate various string constructor with proper syn. programs.

```
class SubStringCons {
    public static void main(String args[]){
    byte ascii[] = {65,66,67,68,69,70};
    String s1 = new String(ascii);
    System.out.println(s1);
    String s2 = new String(ascii,2,3);
    System.out.println(s2);
    }
}
```

Output:

ABCDEF
CDE

2. Using getchars(), extract BMSCE from "Welcome to BMSCE College".

```
public class ExtractSubstring{
    public static void main(String[] args){
    String inputString = "Welcome to Bmsce College";
    int startIndex = 11;
    int endIndex = 16;
    char[] extractedChars = new char[endIndex - startIndex];
    inputString.getChars(startIndex, endIndex, extractedChars, 0);
    String extractedString = new String(extractedChars);
    System.out.println(extractedString);
    }
}
```

Oulfact:

Bmscl.

3. Write a Java program to create a generic class Stack which hold 5 integers & 5 double values.

```
import java.util.ArrayList;
import java.util.List;
public class GenericStack<T>{
    private List<T> stack;
    public GenericStack(){
        this.stack = new ArrayList<>();
}
public void push(T item){
    if(stack.size()<5){
        stack.add(item);
        System.out.println("Pushed: "+item);}
    else{
        System.out.println("Stack is empty. Cannot
        pop elements.");
        return null;
    }
}
public static void main(String[] args){
    GenericStack<Integer> integerStack = new GenericStack<>();
    integerStack.push(1);
    integerStack.push(2);
    integerStack.push(3);
    integerStack.pop();
    integerStack.pop();
}
```

```
GenericStack<Double> doubleStack = new GenericStack<>();
doubleStack.push(1.1);
doubleStack.push(2.2);
doubleStack.push(3.3);
doubleStack.pop();
doubleStack.pop();
}
}
```

Output:

```
Pushed: 1
Pushed: 2
Pushed: 3
Popped: 3
Popped: 2
Pushed: 1.1
Pushed: 2.2
Pushed: 3.3
Popped: 3.3
Popped: 2.2
```

4. Write a Java program to create an abstract class Bird with abstract methods fly() & makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.

```
abstract class Bird {
    abstract void fly();
    abstract void makeSound();
}
class Eagle extends Bird {
    void fly(){
        System.out.println("Eagle is soaring high in
        the sky.");
    }
    void makeSound() {
        System.out.println("Eagle makes a screeching
        sound.");
    }
}
class Hawk extends Bird {
    void fly(){
        System.out.println("Hawk is gliding
        gracefully.");
    }
    void makeSound(){
        System.out.println("Hawk makes a piercing
        cry.");
    }
}
public class BirdTest {
    public static void main(String[] args){
    Eagle eagle = new Eagle();
    Hawk hawk = new Hawk();
    System.out.println("Eagle Actions:");
    eagle.fly();
    eagle.makeSound();
    System.out.println();
    System.out.println("Hawk Actions:");
    hawk.fly();
    hawk.makeSound(); } }
```

Output:

Eagle Actions:

Eagle is soaring high in the sky.
Eagle makes a screeching sound.

Hawk Actions:

Hawk is gliding gracefully.
Hawk makes a piercing cry.

16/11/24