Reference Guide for Stack Tracing Java

2025-05-12

FYI

- Only a single color is required for memory diagrams, different colors are used (and order written into code) for greater clarity in this document

1 Basic Java

```
public static void main(String[] args) {
    int anInt = 10;
    double aDouble = 5.8;
    boolean Boolean = true;
    String aString = "6.3";
    anInt = 20;
}
```

- variable changes result in previous values being crossed out and new ones written in so that progression can be seen

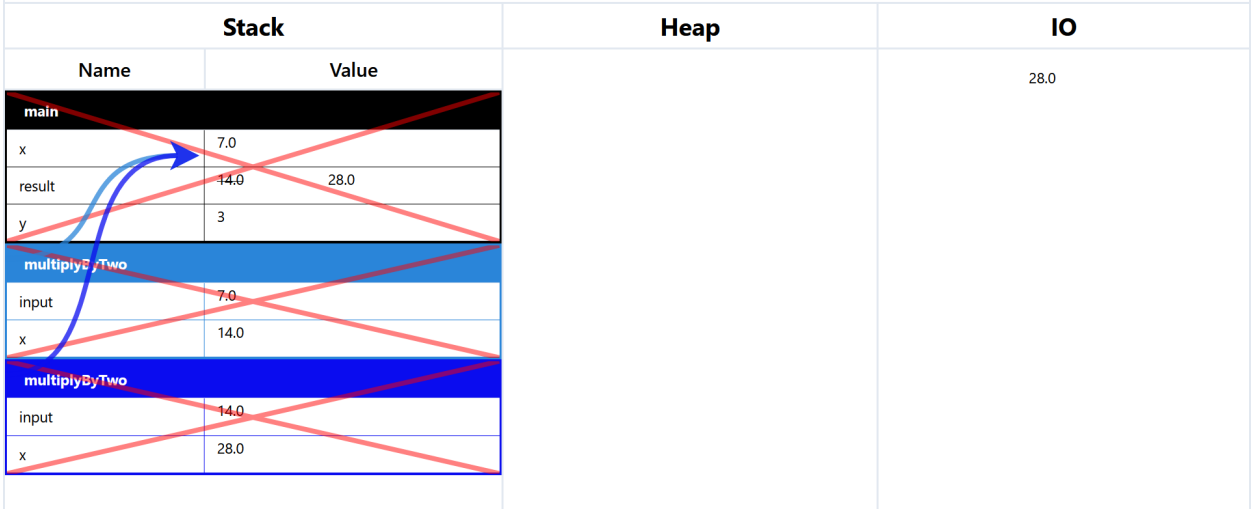
Stack		Heap	IO
Name	Value		
main			
anInt	~~10~~ 20		
aDouble	5.8		
Boolean	true		
aString	"6.3"		

2 Function calls

2.1 Return value

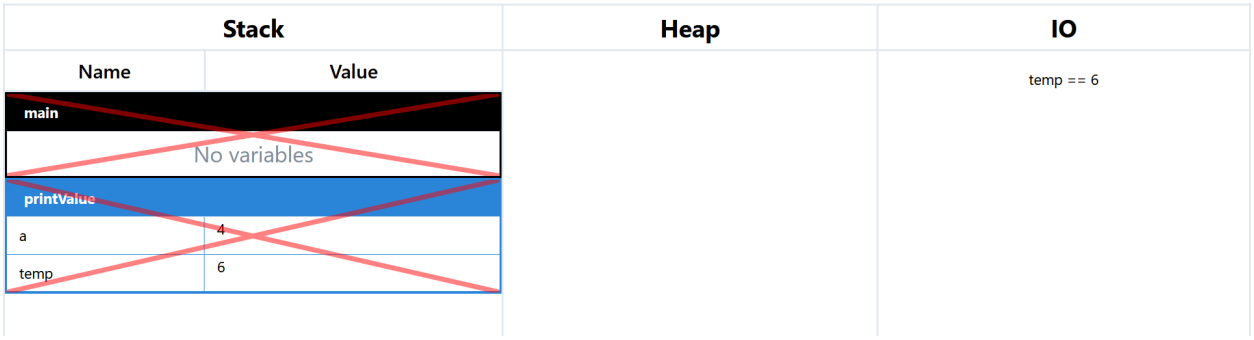
```
public static double multiplyByTwo(double input) {
    double x = input * 2;
    return x;
}

public static void main (String[] args) {
    double x = 7.0;
    double result = multiplyByTwo(x);
    result = multiplyByTwo(result);
    System.out.println(result);
    int y = 3;
}
```

- each function call is put in its own stack frame
- variables created after the function call appear further down the stack
- IO stands for input/output and is where any command line user input or outputs in terms of print statements appear



2.2 No return value

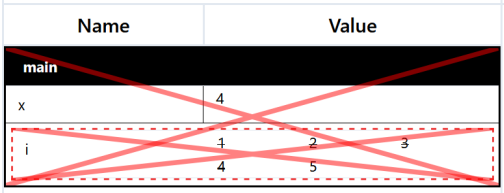
```
public static double printValue(int a) {
    int temp = n+2;
    System.out.println("temp == " + temp);
}

public static void main (String[] args) {
    printValue(4);
}
```



3 For loops

3.1 Basic

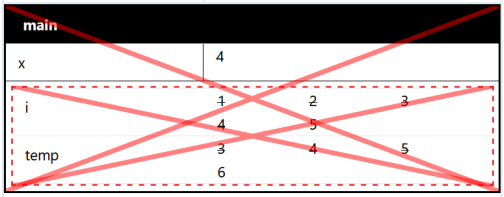
```
public static void main (String[] args) {
    int x = 4;
    for (int i = 1; i < 5; i++) {
        System.out.println("i == " + i);
    }
}
```

Stack	Heap	IO
		i == 1 i == 2 i == 3 i == 4

3.2 Scoped Variable

```
public static void main (String[] args) {
    int x = 4;
    for (int i = 1; i < 5; i++) {
        int temp = i + 2;
    }
}
```

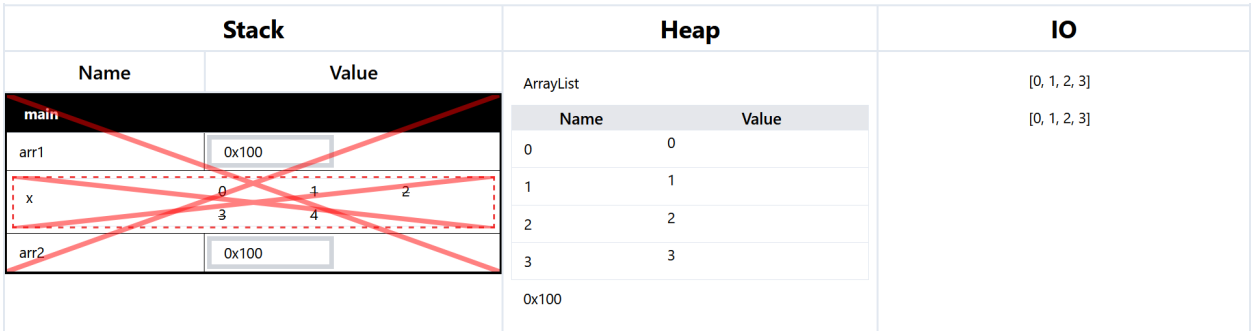
- scoped variables within the loop are crossed out after the loop completes

Stack	Heap	IO
		

4 ArrayLists

```
public static void main (String[] args) {
    ArrayList<Integer> arr1 = new ArrayList<>();
    for (int x = 0; x < 4; x++) {
        arr1.add(x);
    }
    System.out.println(arr1);
    ArrayList<Integer> arr2 = arr1;
    System.out.println(arr2);
}
```

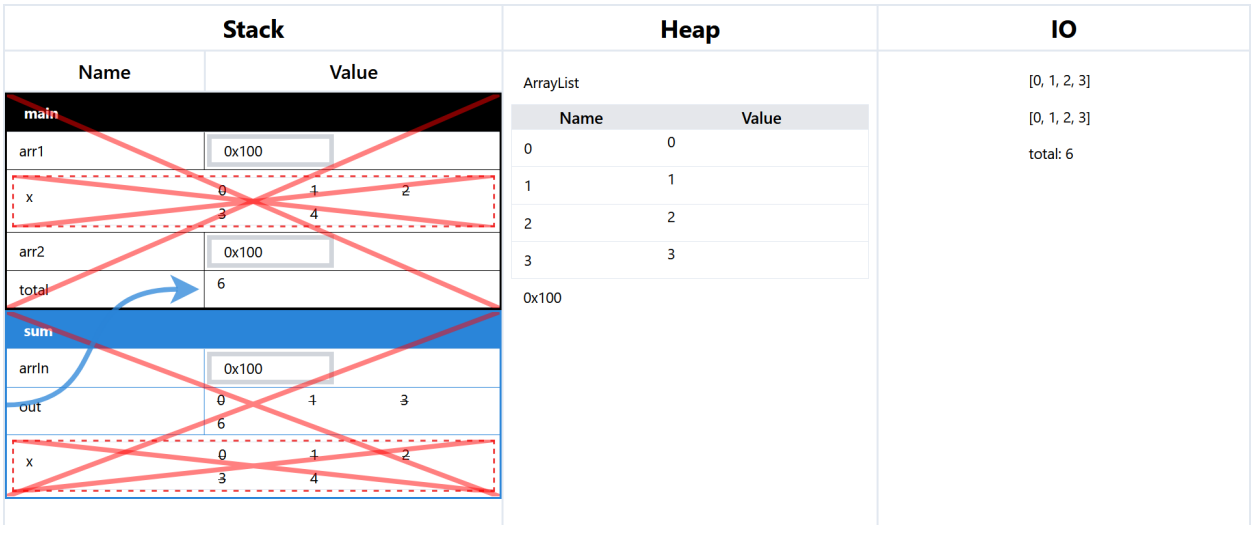
- note that the assignment statement for arr2 assigns the memory address and does not do a deep copy
  - this point is emphasized for newer programmers
- memory addresses all start with 0x to indicate that they are hexadecimal numbers.
  - heap addresses are typically given 3 digit numbers
  - numbers are typically written in decimal as it is easier for students to grasp at first (as they are not familiar with hexadecimal, this detail will be corrected in later courses)
  - numbers are randomly generated and just must agree on the stack and heap



4.1 Passed to functions

```
public static int sum(ArrayList<Integer> arrIn) {
    int out = 0;
    for (int x = 0; x < arrIn.size(); x++) {
        out += arrIn.get(x);
    }
    return out;
}

public static void main (String[] args) {
    ArrayList<Integer> arr1 = new ArrayList<>();
    for (int x = 0; x < 4; x++) {
        arr1.add(x);
    }
    System.out.println(arr1);
    ArrayList<Integer> arr2 = arr1;
    System.out.println(arr2);
    int total = sum(arr1);
    System.out.println("total: " + total);
}
```

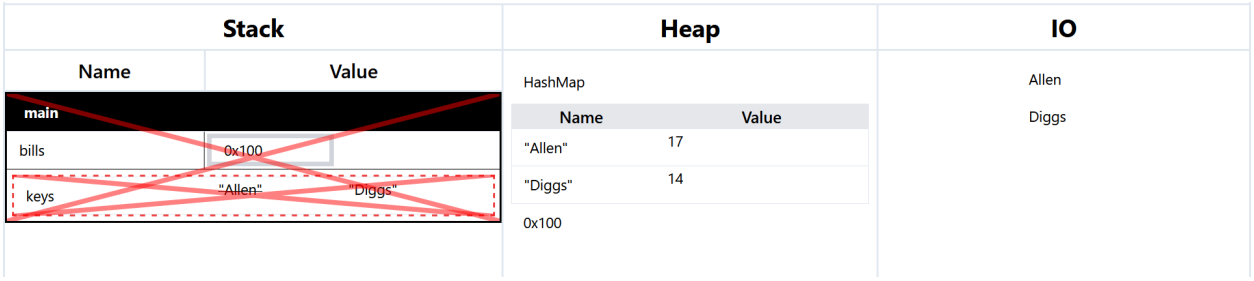
- when passing variables to functions as arguments the value on the stack associated with the variable name is the argument to the function that sets the parameter
- the dashed lines around index indicate that it is a scoped variable that only exists while the loop exists



5 HashMaps

```
public static void main (String[] args) {
    HashMap<String, Integer> bills = new HashMap<>();

    bills.put("Allen", 17);
    bills.put("Diggs", 14);

    for (String keys: bills.keySet()) {
      System.out.println(keys);
    }
}
```

- HashMaps are like dictionaries in python
- we loop through them in a manner more similar to loops in python
- note that the loop still has scoped variables like the previous for loop

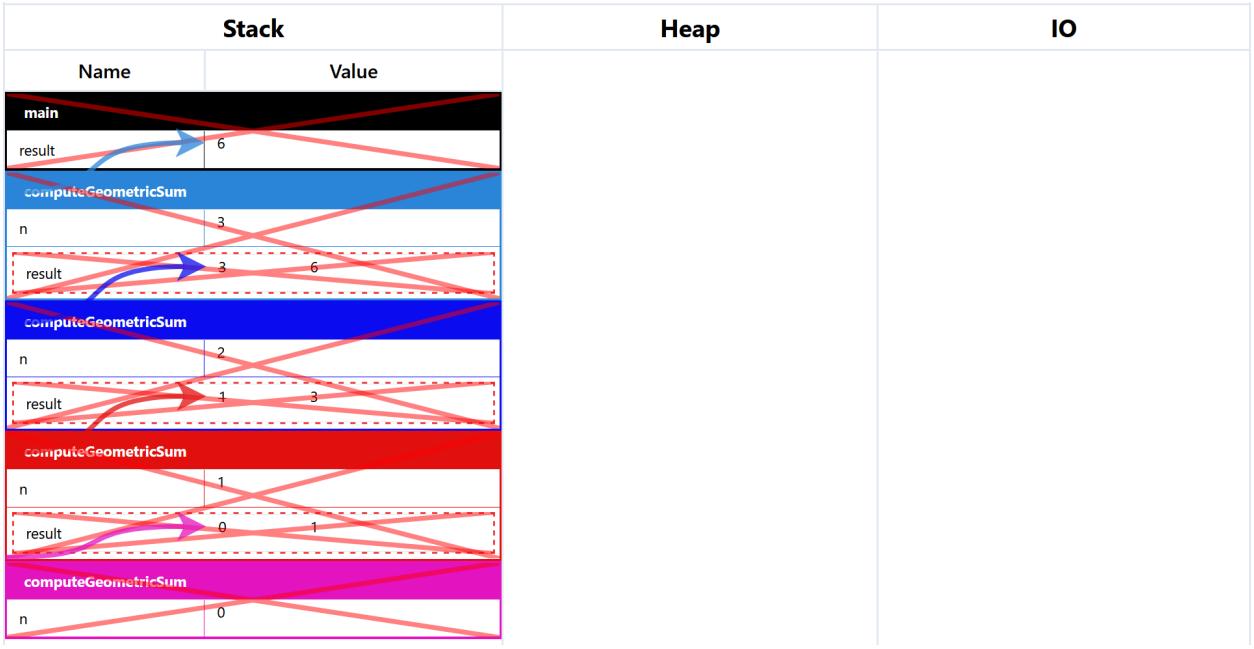


6 Recursion

6.1 Standard Recursion

```
public static int computeGeometricSum(int n) {
    if (n > 0) {
        int result = computeGeometricSum(n - 1);
        result += n;
        return result;
    } else {
        return 0;
    }
}

public static void main (String[] args) {
    int result = computeGeometricSum(3);
}
```

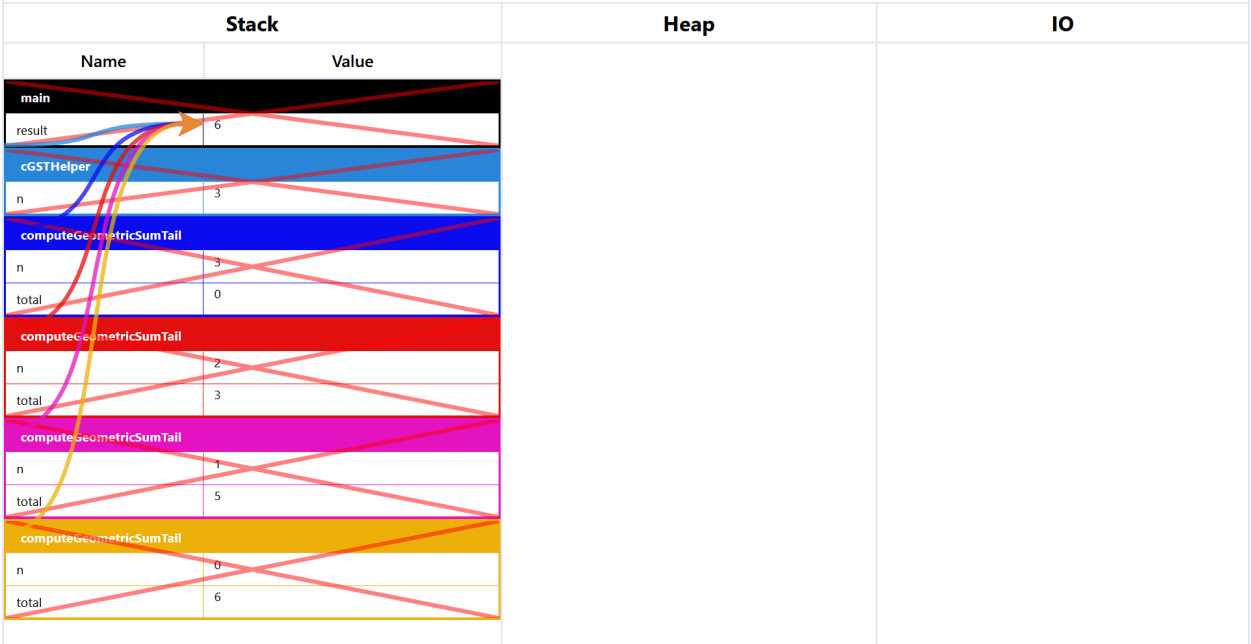
- each new call of cGS is performed in a new color
  - returned values are kept in the color of the method that returns it



6.2 Tail Recursion

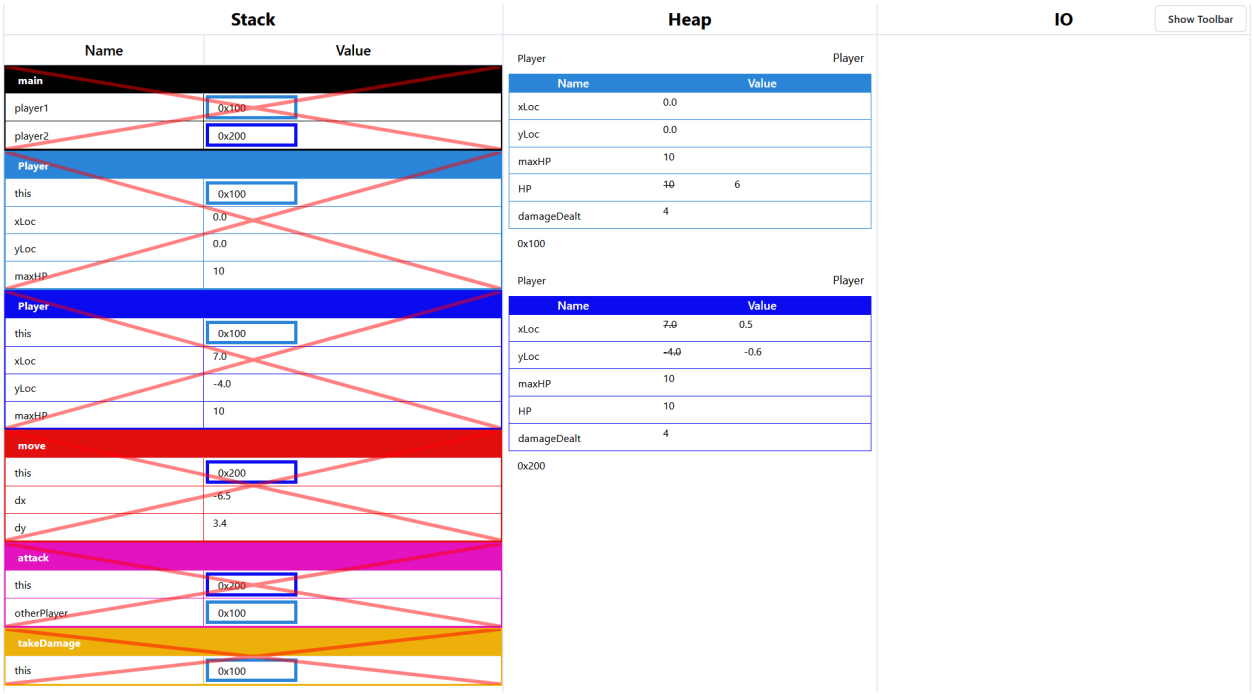
```
public static int computeGeometricSumTail(int n, int total) {
    if (n > 0) {
        return computeGeometricSum(n - 1, total + n);
    } else {
        return total;
    }
}

public static int cGSTHelper(int n) {
    return computeGeometricSumTail(n, 0);
}

public static void main (String[] args) {
    int result = cGSTHelper(3);
}
```

- each new call of cGST is performed in a new color
  - returned values are kept in the color of the method that returns it
- Note that the returns go to the previous function's return and not a variable
  - this is why the memory of a stack frame can be released before the following recursive function call finishes



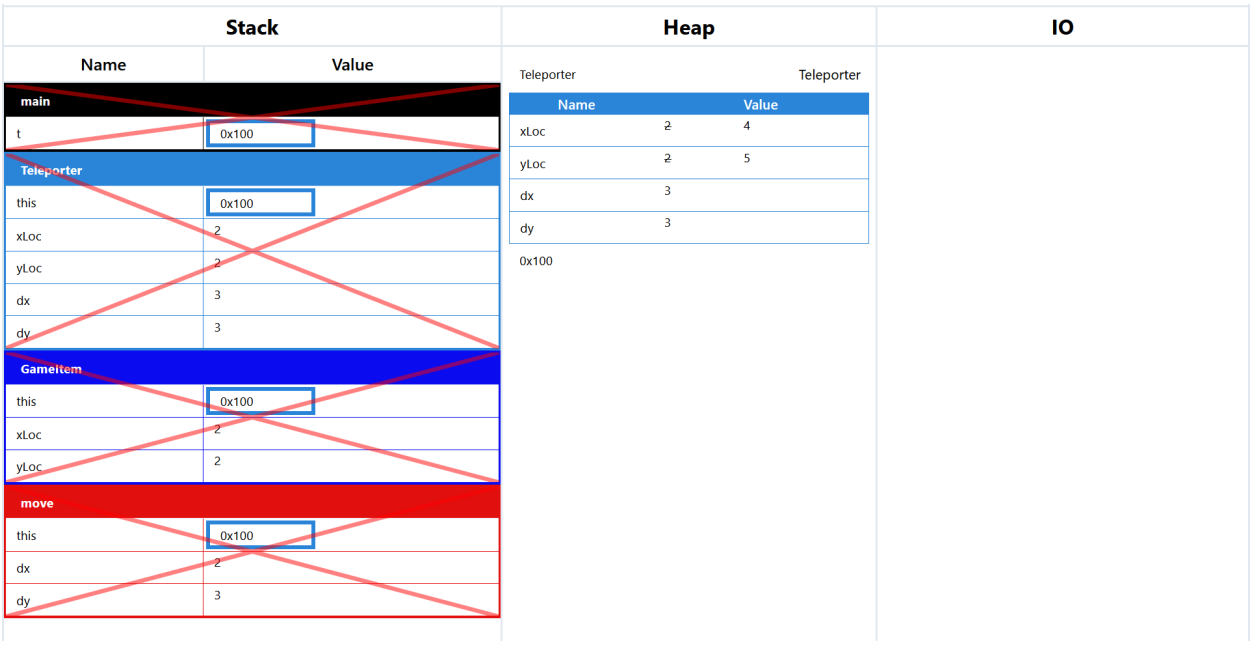
7 Classes

```
public class Player {
    private double xLoc;
    private double yLoc;
    private int maxHP;
    private int HP;
    private int damageDealt;

    public Player(double xLoc, double yLoc, int maxHP) {
        this.xLoc = xLoc;
        this.yLoc = yLoc;
        this.maxHP = maxHP;
        this.HP = maxHP;
        this.damageDealt = 4;
    }
    public int getHP() {
        return this.HP;
    }
    public void takeDamage(int damage) {
        this.HP -= damage;
    }
    public void attack(Player otherPlayer) {
        otherPlayer.takeDamage(this.damageDealt);
    }
    public void move(double dx, double dy) {
        this.xLoc += dx;
        this.yLoc += dy;
    }

    public static void main(String[] args) {
        Player player1 = new Player(0.0, 0.0, 10);
        Player player2 = new Player(7.0, -4.0, 10);
        player2.move(-6.5, 3.4);
        player2.attack(player1);
    }
}
```


Stack
Name
Value
main
player1
0x100
player2
0x200
Player
this
0x100
xLoc
0.0
yLoc
0.0
maxHP
10
Player
this
0x100
xLoc
7.0
yLoc
-4.0
maxHP
10
move
this
0x200
dx
-6.5
dy
3.4
attack
this
0x200
otherPlayer
0x100
takeDamage
this
0x100
Heap
Player
Player
Name
Value
xLoc
0.0
yLoc
0.0
maxHP
10
HP
10
6
damageDealt
4
0x100
Player
Player
Name
Value
xLoc
7.0
0.5
yLoc
-4.0
-0.6
maxHP
10
HP
10
damageDealt
4
0x200
IO
Show Toolbar

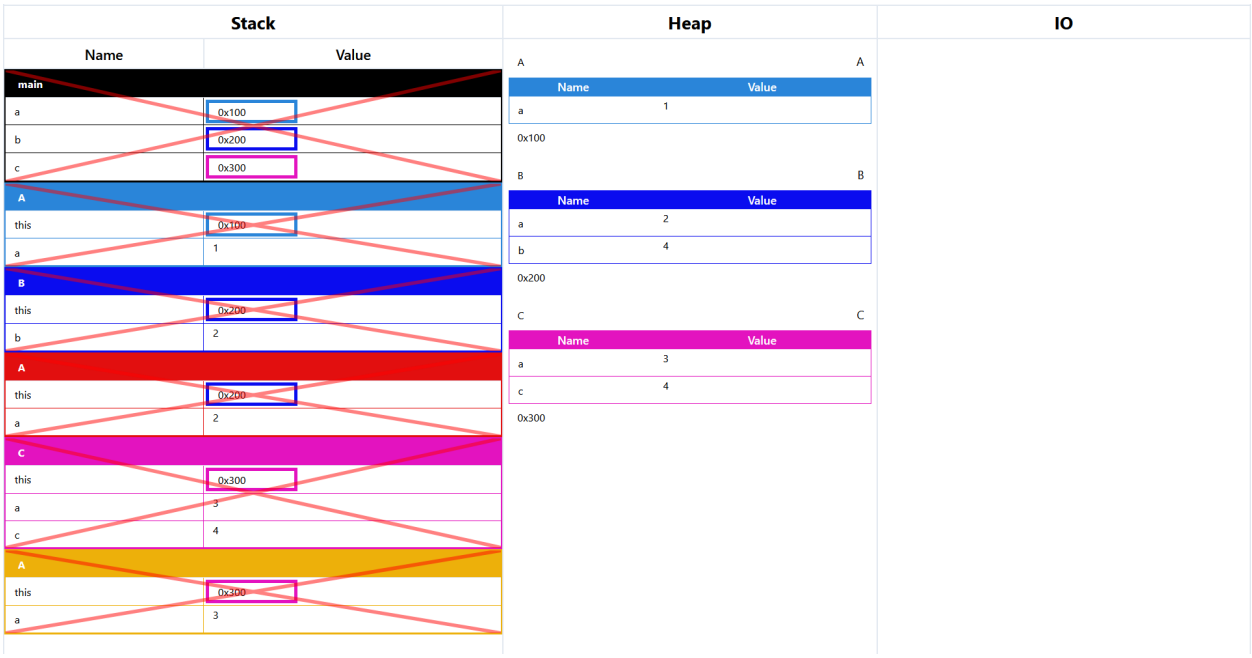
8 Inheritance

```
public class GameItem {
    private double xLoc;
    private double yLoc;

    public GameItem(double xLoc, double yLoc) {
        this.xLoc = xLoc;
        this.yLoc = yLoc;
    }
    public void move(double dx, double dy) {
        this.xLoc += dx;
        this.yLoc += dy;
    }
}

public class Teleporter extends GameItem {
    private double dx;
    private double dy;

    public Teleporter(double xLoc, double yLoc, double dx, double dy) {
        super(xLoc, yLoc);
        this.dx = dx;
        this.dy = dy;
    }

    public static void main(String[] args) {
        Teleporter t = new Teleporter(2, 2, 3, 3);
        t.move(2, 3);
    }
}
```


Stack
Name
Value
main
t
0x100
Teleporter
this
0x100
xLoc
2
yLoc
2
dx
3
dy
3
GameItem
this
0x100
xLoc
2
yLoc
2
move
this
0x100
dx
2
dy
3
Heap
Teleporter
Teleporter
Name
Value
xLoc
2
4
yLoc
2
5
dx
3
dy
3
0x100
IO

9 Polymorphism

```
public class A {
    protected int a;

    public A(int a) {
        this.a = a;
    }
}

public class B extends A{
    private int b;

    public B(int b) {
        super(b);
        this.b = b*2;
    }
}

public class C extends A{
    private int c;

    public C(int a, int c) {
        super(a);
        this.c = c;
    }
}

public class RunABC {
    public static void main(String[] args) {
        A a = new A(1);
        A b = new B(2);
        A c = new C(3, 4);
    }
}
```


Stack
Name
Value
main
a
0x100
b
0x200
c
0x300
A
this
0x100
a
1
B
this
0x200
b
2
A
this
0x200
a
2
C
this
0x300
a
3
c
4
A
this
0x300
a
3
Heap
A
A
Name
Value
a
1
0x100
B
B
Name
Value
a
2
b
4
0x200
C
C
Name
Value
a
3
c
4
0x300
IO