Reference Guide for Stack Tracing Java

February 20, 2024

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 variables

```java
public static void main(String[] args) {
  int anInt = 10;
  double aDouble = 5.8;
  boolean aBoolean = 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**		
Global Variables		CREATE HEAP OBJECT	CREATE IO LINE
main			
anInt	~~10~~ 20		
aFloat	5.8		
aBoolean	true		
aString	"6.3"		
CREATE GLOBAL VARIABLE			
CREATE STACK FRAME			

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

- in/out stands for input/output and is where any command line user input or outputs in terms of print statements appear

Stack		Heap	IO
Name	**Value**		
Global Variables			28.0 ✕
main ♻ ← ✕		CREATE HEAP OBJECT	CREATE IO LINE
✕ x	7.0 ⊖		
✕ result	14.0 28.0 ⊖		
⊕ ☐ ✕ y	3 ⊖		
multiplyByTwo ♻ ✕			
✕ This stack frame has been crossed out Hover to view	⊖		
⊕ ☐ ✕ x	14.0 ⊖		
multiplyByTwo ♻ ✕			
✕ This stack frame has been crossed out Hover to view	⊖		
⊕ ☐ ✕ x	28.0 ⊖		

2.2 No return value

```
public static void printValue(int n): {
    int temp=n+2
    System.out.println("temp == " + temp)
}

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

Stack		Heap	IO
Name	**Value**		
Global Variables			temp == 6 ⊗
main ♻ ← ✕		CREATE HEAP OBJECT	
+			CREATE IO LINE
printValue ♻ ← ✕			
⊗ This stack frame has been crossed out Hover to view ⊖			
⊕ ▢ ⊗ temp	6 ⊖		

3 For loops

3.1 basic

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

Stack		Heap	IO
Name	**Value**		

Global Variables			

Stack section:
- main ♻ ← X
- (+) (□) (X)   x   4   (−)
- (+) (X)   i   1̶ 2̶ 3̶ 4̶ 5   (−)

Heap: CREATE HEAP OBJECT

IO:
- i == 1 (X)
- i == 2 (X)
- i == 3 (X)
- i == 4 (X)

3.2 scoped variable

```java
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
Name	**Value**		

Global Variables			

Stack section:
- main ♻ ← X
- (+) (□) (X)   x   4   (−)
- (X)   i   1̶ 2̶ 3̶ 4̶ 5   (−)
- (+) (X)   temp   3̶ 4̶ 5̶ 6   (−)

Heap: CREATE HEAP OBJECT

IO: CREATE IO LINE

4 ArrayLists

```java
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

Stack		Heap		IO
Name	**Value**			
Global Variables		ArrayList No parent ⌄ ⊗		[0, 1, 2, 3] ⊗
main ♻ ← ✕		**Name**	**Value**	[0, 1, 2, 3] ⊗
⊗ arr1	`0x002` ⊖	⊗ 0	0 ⊖	**CREATE IO LINE**
⊕ ⊗ x	0̶ 1̶ 2̶ 3̶ **4** ⊖	⊗ 1	1 ⊖	
⊕ ⊡ ⊗ arr2	`0x002` ⊖	⊗ 2	2 ⊖	
CREATE GLOBAL VARIABLE		⊕ ⊗ 3	3 ⊖	
		0x002		

4.1 Passed to functions

```java
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

- HashMaps are like dictionaries in python

- we loop through them in a manner more similar to loops in python

-
```
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);
    }
}
```

Stack		Heap		IO

Name	Value
Global Variables	
main ♻ ← ✕	
➕ ⬜ ✕ bills	0x002 ➖
➕ ✕ keys	~~"Allen"~~ "Diggs" ➖

HashMap [No parent ⌄] ✕

Name	Value
✕ "Allen"	17 ➖
➕ ✕ "Diggs"	14 ➖

0x002

IO:
Allen ✕
Diggs ✕

CREATE IO LINE

- note that the loop still has scoped variables like the previous for loop

7

6 Recursion

6.1 standard recursion

```java
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

Stack		Heap	IO
Name	**Value**		

Global Variables

main   ♻ ← ✕
⊕ ⊡ ➤ result | 6 | ⊖

CREATE HEAP OBJECT

CREATE IO LINE

computeGeometricSum   ♻ ⊗ ✕
✕ This stack frame has been crossed out ⊖
   Hover to view
⊕ ⊡ ➤ result | 3̶ 6 | ⊖

computeGeometricSum   ♻ ⊗ ✕
✕ This stack frame has been crossed out ⊖
   Hover to view
⊕ ⊡ ➤ result | 1̶ 3 | ⊖

computeGeometricSum   ♻ ⊗ ✕
✕ This stack frame has been crossed out ⊖
   Hover to view
⊕ ⊡ ➤ result | 0̶ 1 | ⊖

computeGeometricSum   ♻ ⊗ ✕
This stack frame has been crossed out
⊕ ⊡ ✕   Hover to view   ⊖

6.2 Tail recursion

```java
public static int computeGeometricSumTail(int n,int total){
    if (n>0){
        return computeGeometricSumTail(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 functions return and not a variable

  - this is why the memory of a stack frame can be released before the following recursive function call finishes

Stack		Heap	IO
Name	**Value**		
Global Variables			
main ♻ ← ✕		CREATE HEAP OBJECT	CREATE IO LINE
➕ ◻ ▲ result	value-1 ➖		
cGSTHelper ♻ ✂ ✕			
➕ ◻ ✕ This stack frame has been crossed out Hover to view ➖			
computeGeometricSum ♻ ✂ ✕			
✕ This stack frame has been crossed out Hover to view ➖			
➕ ◻ ✕ total	0 ➖		
computeGeometricSum ♻ ✂ ✕			
✕ This stack frame has been crossed out Hover to view ➖			
➕ ◻ ✕ total	3 ➖		
computeGeometricSum ♻ ✂ ✕			
✕ This stack frame has been crossed out Hover to view ➖			
➕ ◻ ✕ total	5 ➖		
computeGeometricSum ♻ ✂ ✕			
✕ This stack frame has been crossed out Hover to view ➖			
➕ ◻ ✕ total	6 ➖		

7 Classes

```java
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		Heap	IO

Stack

Name	Value

Global Variables

main ♻ ← ✕

| ✕ | player1 | 0x002 | ⊖ |
| ✕ (+ ▢ ✕) | player2 | 0x003 | ⊖ |

Player ♻ ⌨ ✕

| ✕ | this | 0x002 | ⊖ |

This stack frame has been crossed out
Hover to view

| ✕ | yLoc | 0.0 | ⊖ |
| (+ ▢ ✕) | maxHP | 10 | ⊖ |

Player ♻ ⌨ ✕

| ✕ | this | 0x003 | ⊖ |

This stack frame has been crossed out
Hover to view

| ✕ | yLoc | -4.0 | ⊖ |
| (+ ▢ ✕) | maxHP | 10 | ⊖ |

move ♻ ← ✕

| ✕ | this | 0x003 | ⊖ |

This stack frame has been crossed out
Hover to view

| ✕ | dx | | ⊖ |
| (+ ▢ ✕) | dy | 3.4 | ⊖ |

attack ♻ ← ✕

This stack frame has been crossed out
Hover to view

| ✕ | otherPlayer | 0x002 | ⊖ |
| (+ ▢ ✕) | | | |

takeDamage ♻ ← ✕

This stack frame has been crossed out
Hover to view

| ✕ | damage | 4 | ⊖ |
| (+ ▢ ✕) | | | |

Heap

Player [Player ▾] ✕

Name	Value
✕ xLoc	0.0 ⊖
✕ yLoc	0.0 ⊖
✕ maxHP	10 ⊖
✕ HP	~~10~~ 6 ⊖
(+ ✕) damageDealt	4 ⊖

0x002

Player [Player ▾] ✕

Name	Value
✕ xLoc	~~7.0~~ .5 ⊖
✕ yLoc	~~-4.0~~ -.6 ⊖
✕ maxHP	10 ⊖
✕ HP	10 ⊖
(+ ✕) damageDealt	4 ⊖

0x003

CREATE HEAP OBJECT

IO

CREATE IO LINE

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		Heap		IO

Stack

Name	Value
Global Variables	
main	⟳ ← X
(+) (□) (X)　　　　t	[0x002]　　(−)

teleporter ⟳ ⚏ X
(X)	this	[0x002]	(−)
(X)	xLoc	2	(−)
This stack frame has been crossed out			
(X)	yLoc	Hover to view	(−)
(X)	dx	3	(−)
(+) (□) (X)	dy	3	(−)

GameItem ⟳ ← X
(X)	this	[0x002]	(−)
This stack frame has been crossed out			
(X)	xLoc	Hover to view	(−)
(+) (□) (X)	yLoc	2	(−)

move ⟳ ← X
(X)	this	[0x002]	(−)
This stack frame has been crossed out			
(X)	dx	Hover to view	(−)
(+) (□) (X)	dy	3	(−)

Heap

Teleporter [Teleporter ▾] (X)

Name	Value	
(X)　　xLoc	2　4	(−)
(X)　　yLoc	2　5	(−)
(X)　　dx	3	(−)
(+)(X)　dy	3	(−)

0x002

[CREATE HEAP OBJECT]

IO

[CREATE IO LINE]

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);
    }
}
```

- Note that polymorphism loooks no different than regular inheritance tracing because we do not include data type in our memory diagram

Stack		Heap	IO

Stack

Name	Value
Global Variables	
main	♻ ← ✕
✕ a	0x002 ⊖
✕ b	0x003 ⊖
⊕ ☐ ✕ c	0x004 ⊖

A — ♻ ⊠ ✕
✕ This stack frame has been crossed out Hover to view ⊖
⊕ ☐ ✕ a 1 ⊖

B — ♻ ⊠ ✕
✕ This stack frame has been crossed out Hover to view ⊖
⊕ ☐ ✕ b 2 ⊖

A — ♻ ← ✕
✕ This stack frame has been crossed out Hover to view ⊖
⊕ ☐ ✕ a 2 ⊖

C — ♻ ⊠ ✕
✕ this 0x004
✕ a This stack frame has been crossed out Hover to view ⊖
⊕ ☐ ✕ c 4 ⊖

A — ♻ ← ✕
✕ This stack frame has been crossed out Hover to view ⊖
⊕ ☐ ✕ a 3 ⊖

Heap

A [A ▾] ✕

Name	Value
⊕ ✕ a	1 ⊖

0x002

B [B ▾] ✕

Name	Value
✕ a	2 ⊖
⊕ ✕ b	4 ⊖

0x003

C [C ▾] ✕

Name	Value
✕ a	3 ⊖
⊕ ✕ c	4 ⊖

0x004

CREATE HEAP OBJECT

IO

CREATE IO LINE