HKN CS61B Midterm 1 Review

Joey Moghadam
Eric Shen
Evan Ye
Varun Naik

Stack & Heap

```
class Ref1{
    public static void main(String[] args){
        int lol = 5;
        int wat = 10;
        lol = wat;
        lol = lol + 1;

        System.out.println(wat);
    }
}
```

What does the `main` method print out?

10

Stack & Heap

```
class Ref2{
    public static void main(String[] args){
        String a = "Cookies";
        String b = "Pizza rolls";
        String c = a;
        a = b;

        System.out.println(c);
    }
}
```

What does the `main` method print out?

Cookies

Stack & Heap

```
Burrito yum = new Burrito();
Food f = yum;
boolean check = (f == yum);
System.out.println(check);
```

What does the following print?

true

Stack & Heap

```
class Stack{
    public static void change1(int[] i){
        change2(i);
        i[0] = i[0] + 100;
    }

    public static void change2(int[] i){
        int[] j = {0, 0, 0};
        i = j;
    }

    public static void main(String[] args){
        int[] arr = {1, 2, 3};
        change1(arr);
        System.out.println(arr[0]);
    }
}
```

What does the `main` method print out?

101

Stack-Heap

Draw a stack-heap diagram for the code below.

```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap

Draw a stack-heap diagram for the code below.

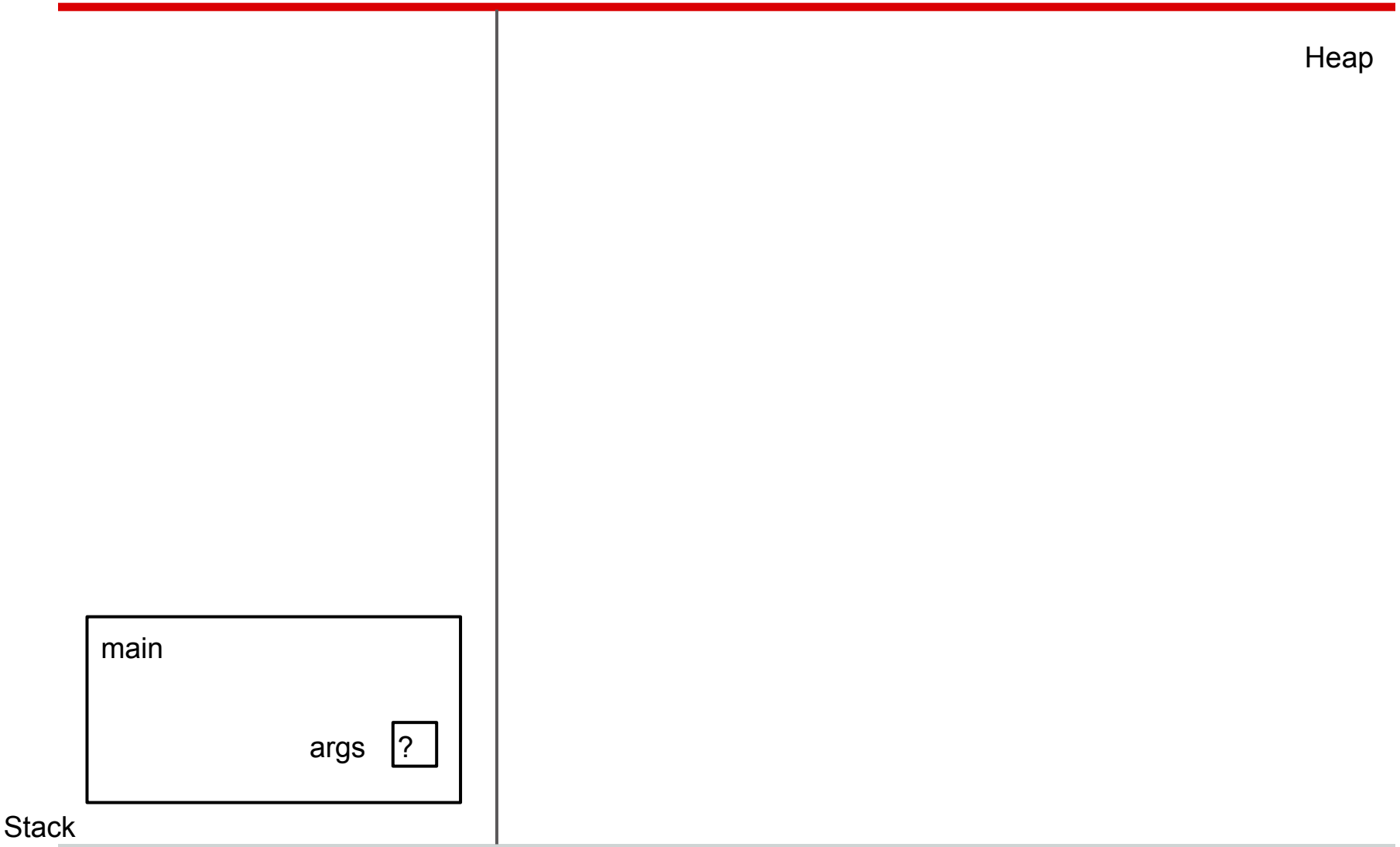
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap



Stack-Heap

Draw a stack-heap diagram for the code below.

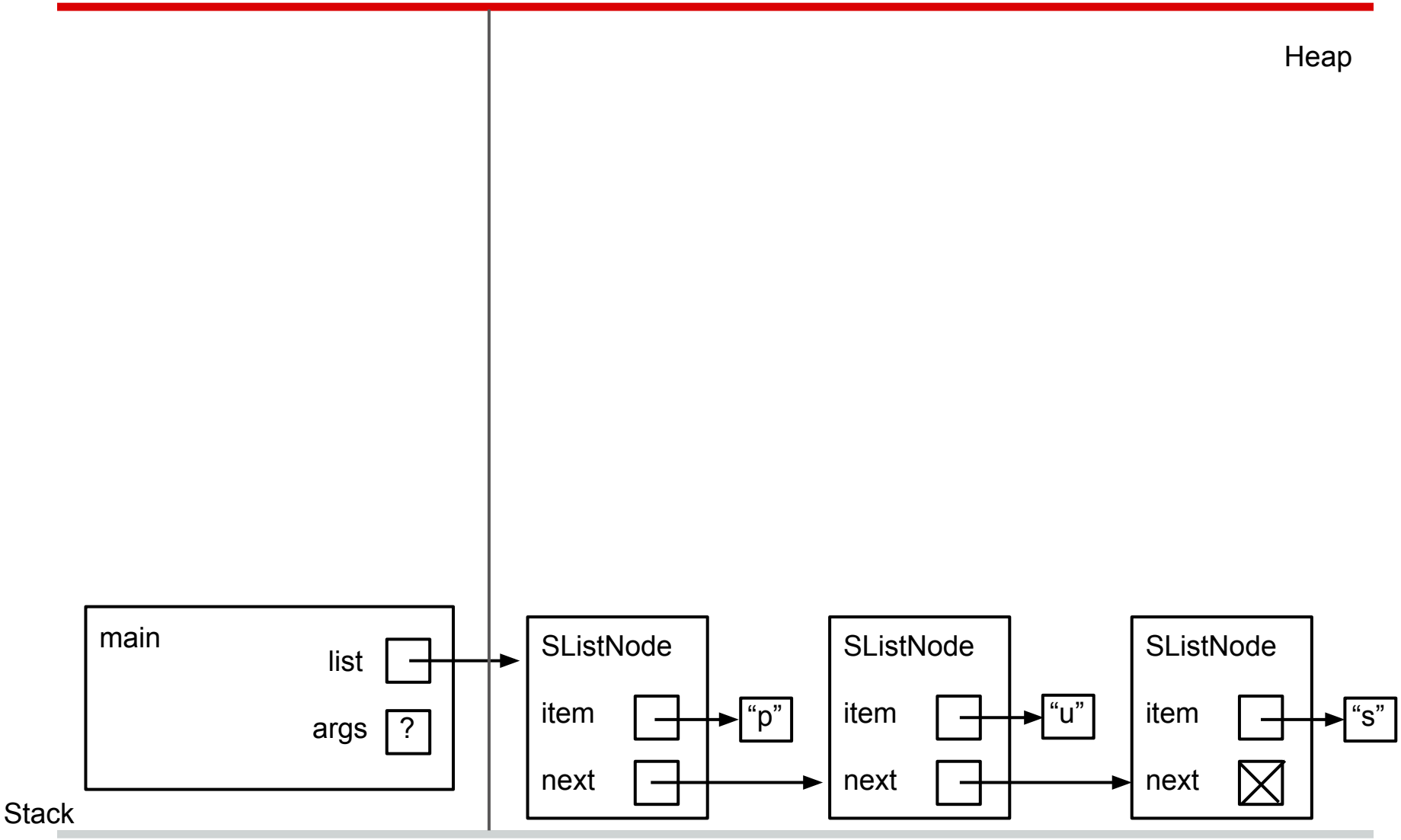
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

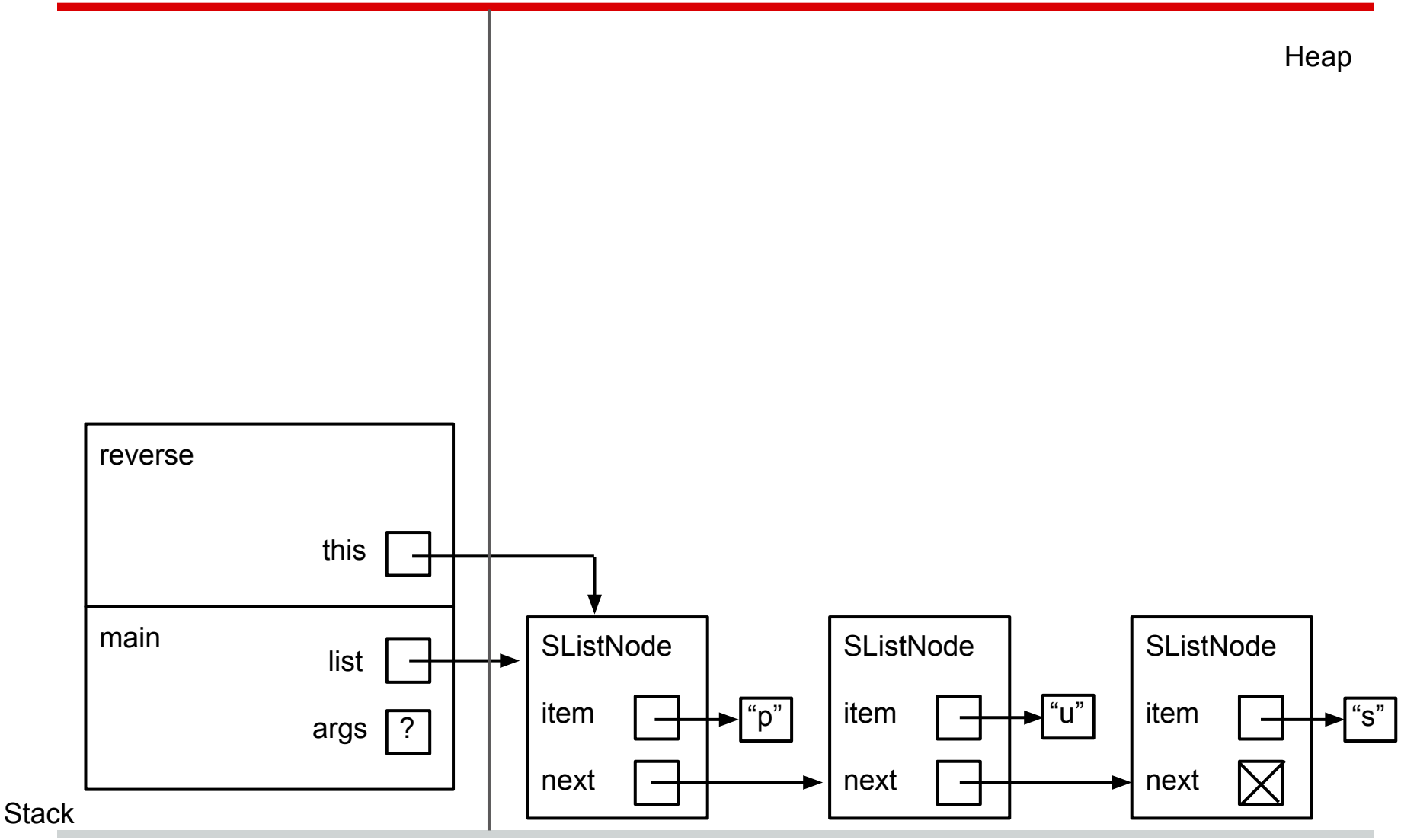
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

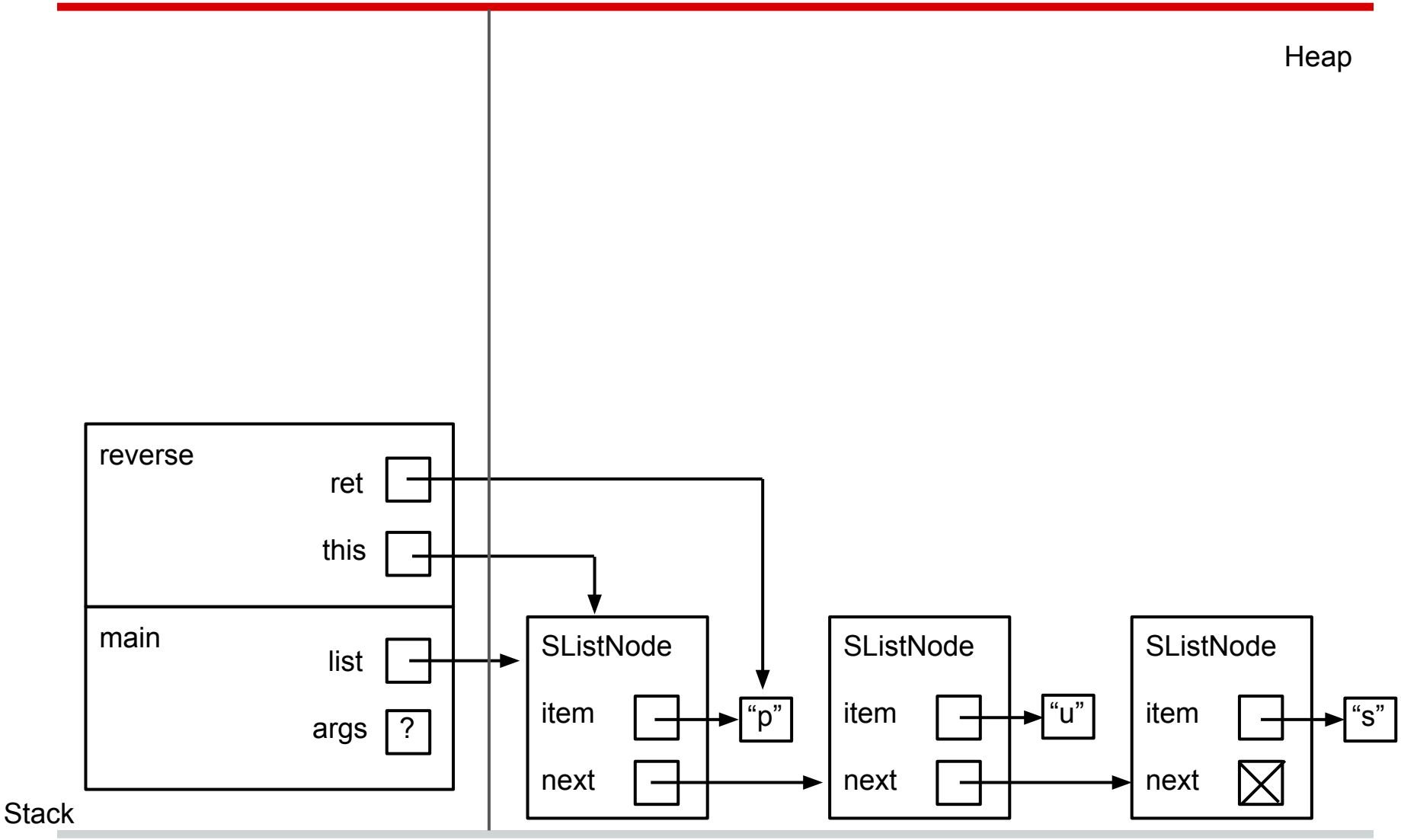
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
ret
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

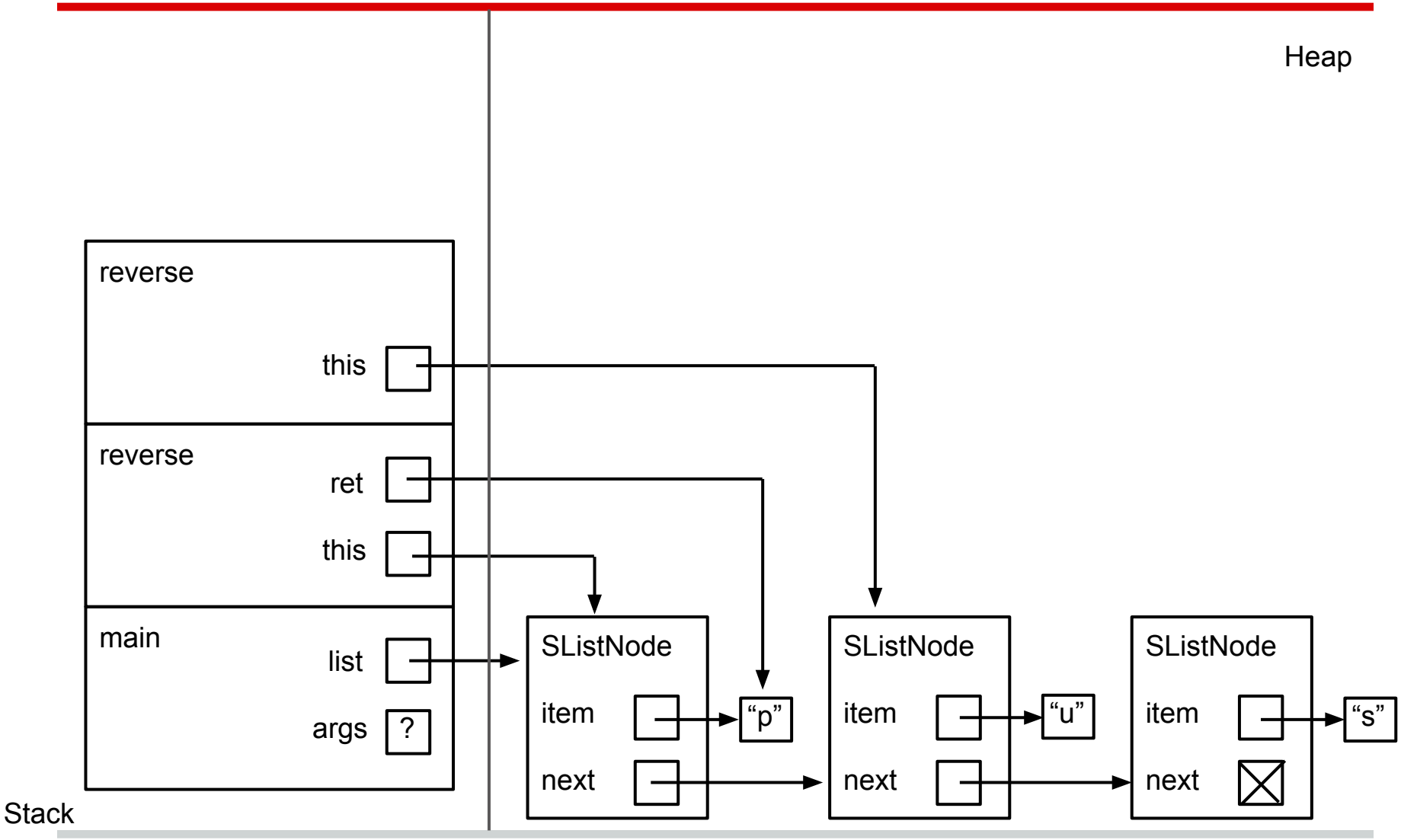
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
this
reverse
ret
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

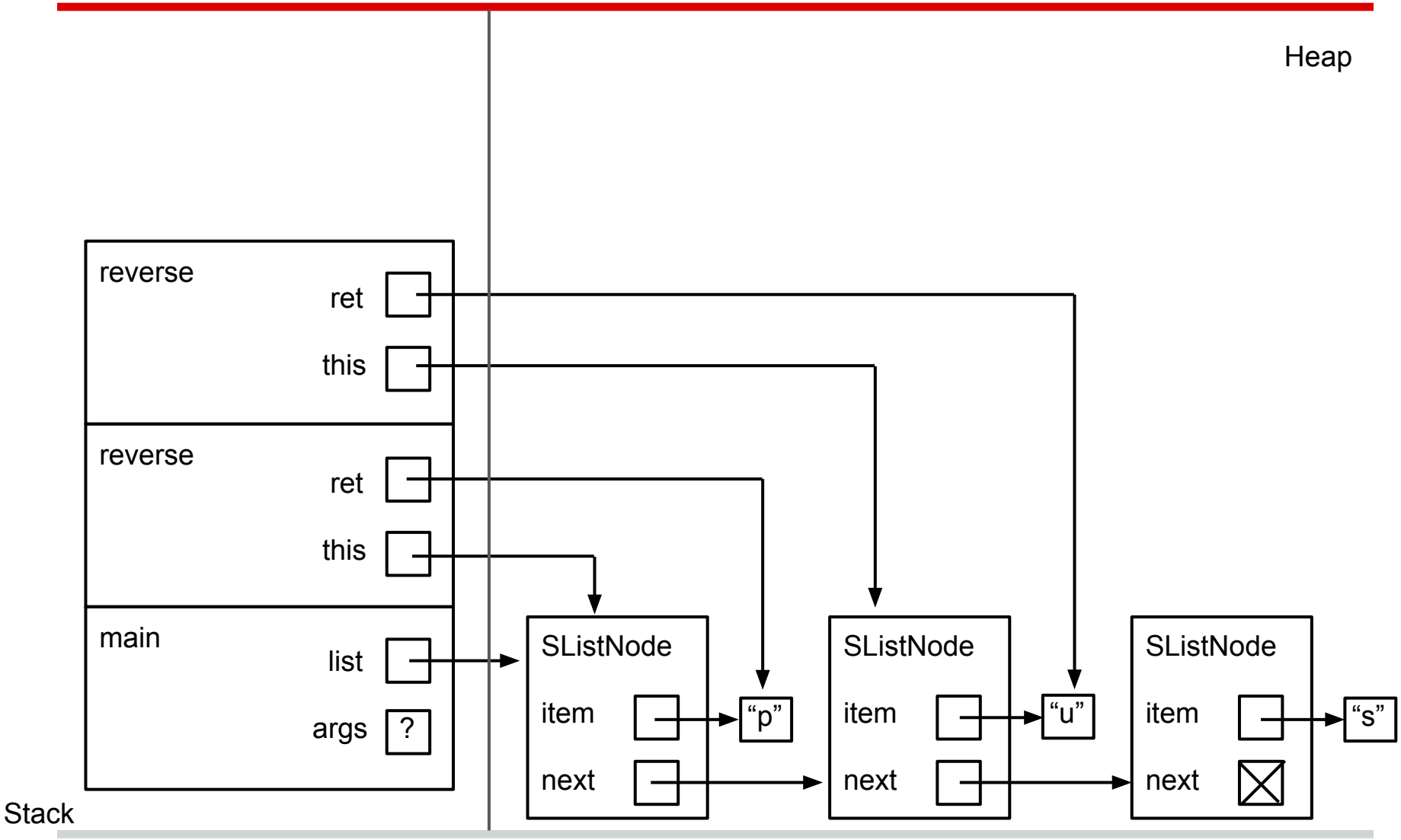
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
ret
this
reverse
ret
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

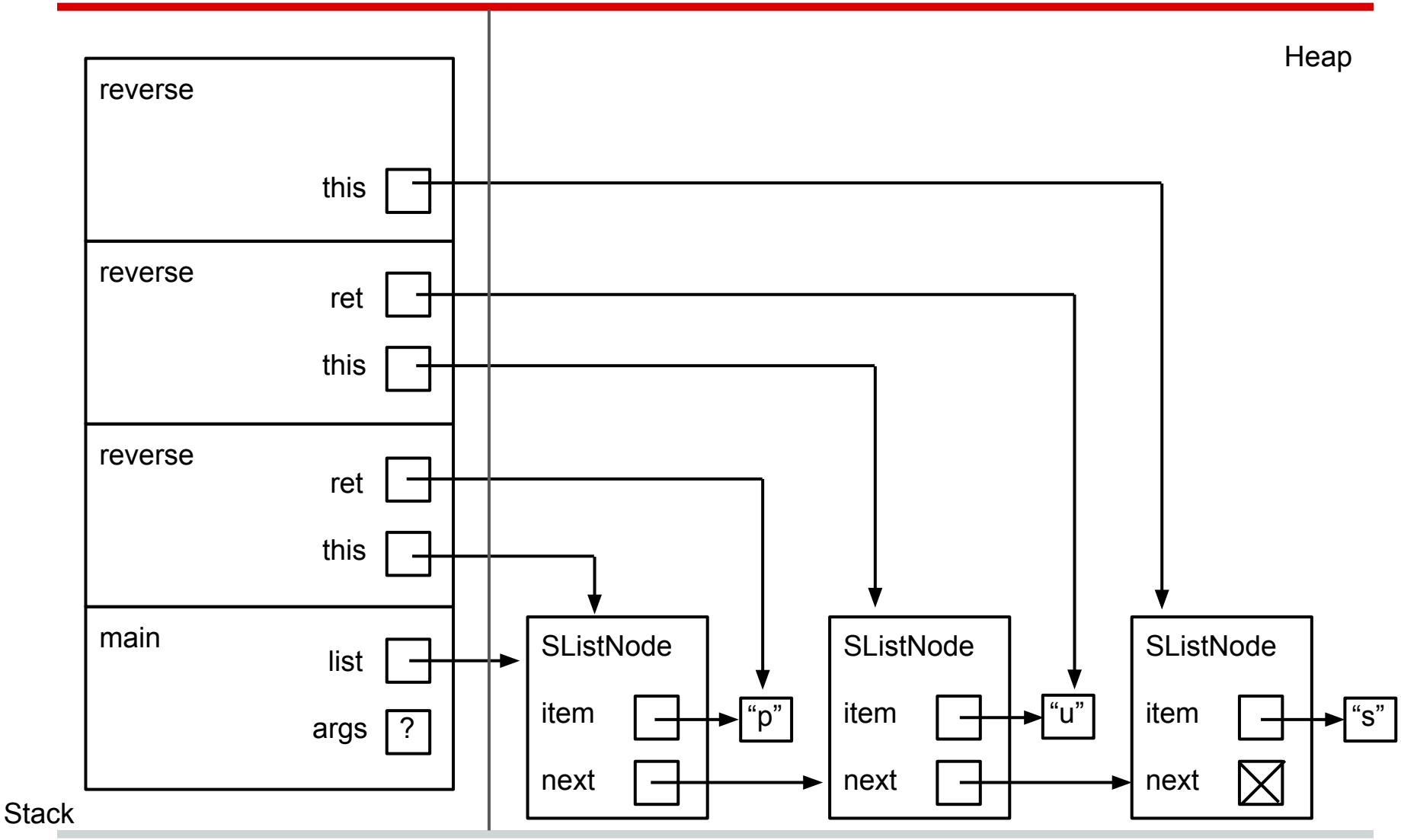
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
this
reverse
ret
this
reverse
ret
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Stack-Heap

Draw a stack-heap diagram for the code below.

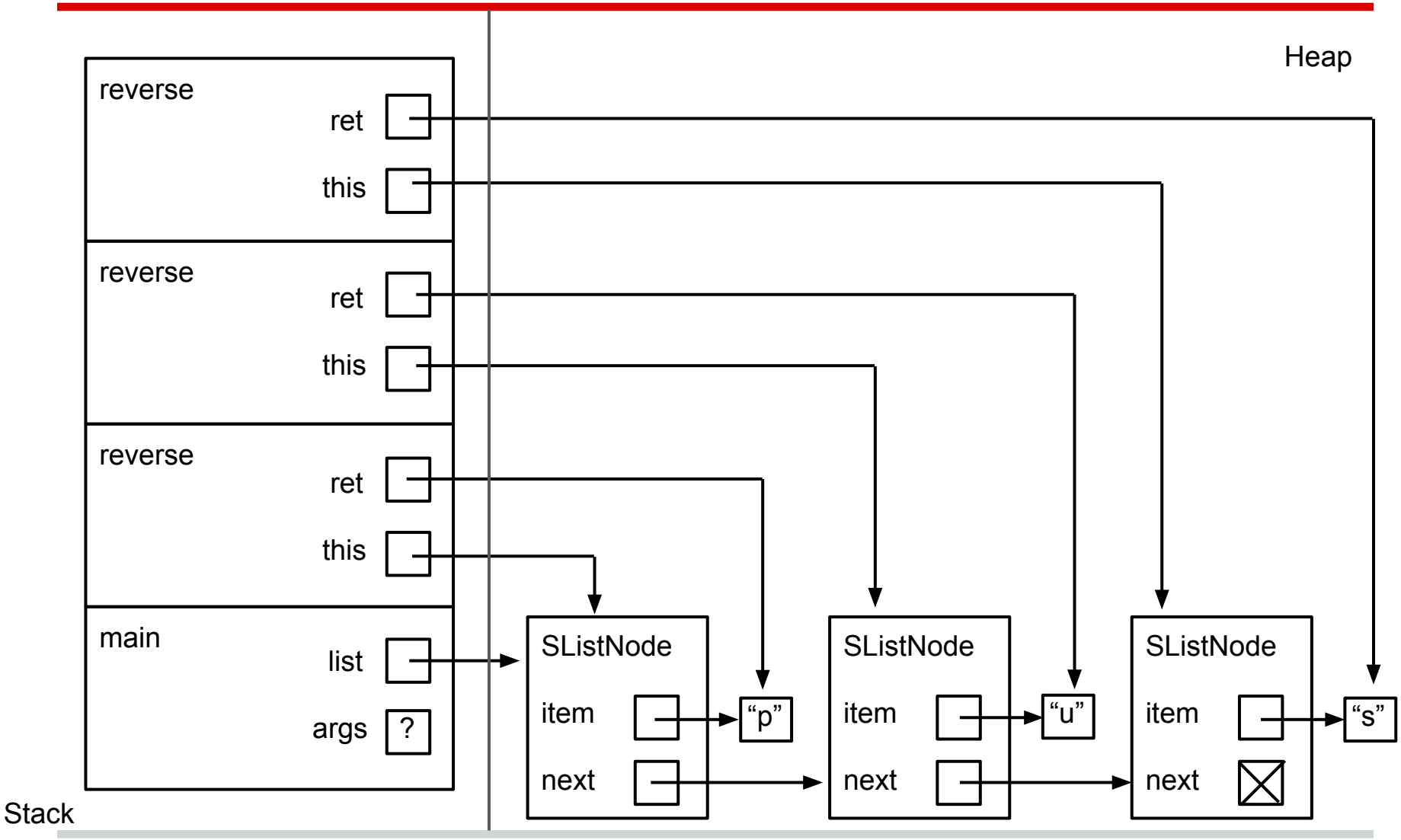
```
public class SListNode {
    public String item;
    public SListNode next;

    public SListNode(String s, SListNode node) {
        item = s;
        next = node;
    }

    public String reverse() {
        String ret = this.item;
        if(this.next == null) {
            //Draw the stack-heap diagram at this point in time
            return item;
        }
        ret = next.reverse() + ret;
        return ret;
    }

    public static void main(String args[]) {
        SListNode list = new SListNode("p", new SListNode("u", new SListNode("s", null)));
        list.reverse();
    }
}
```

Stack-Heap


Heap
reverse
ret
this
reverse
ret
this
reverse
ret
this
main
list
args
?
SListNode
item
next
"p"
SListNode
item
next
"u"
SListNode
item
next
"s"
Stack

Doubly Linked Lists!

```
public class DList {
 protected DListNode head;
 protected DListNode tail;

  public void moveToEnd() {
    // Moves the first node of the list to the end
    /* YOUR CODE HERE */
  }
}
```

Fill in `moveToEnd()`, a method to move the head of the DList to its tail.

Assume you have access to `next` and `prev` attributes.

Doubly Linked Lists!

```
public class DList {
 protected DListNode head;
 protected DListNode tail;

  public void moveToEnd() {
    // Moves the first node of the list to the end
    if(head != null) {
      tail.next = head;
      head.prev = tail;
      tail = head;
      head = head.next;
      tail.next = null;
      head.prev = null;
    }
  }
}
```

Try drawing the pointers!

Singly Linked Lists!

```
public void trim(int min, int max, SList list) {
  /* YOUR CODE HERE */
}
```

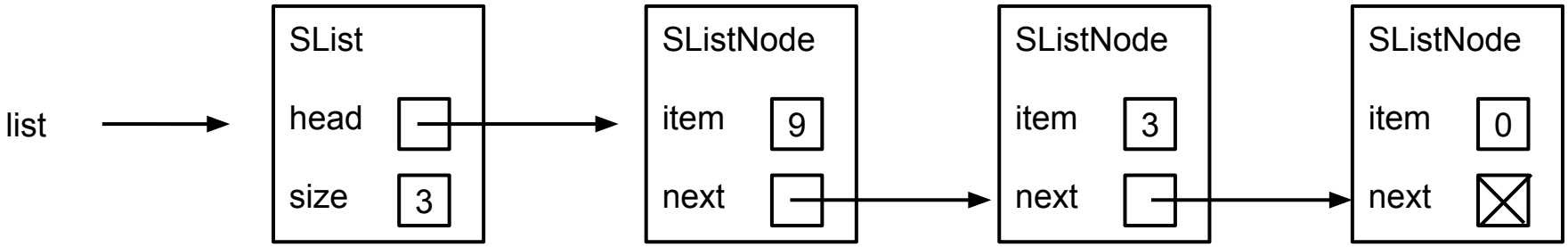
Write a method `trim` that will **destructively** remove elements from the input singly-linked list which have a value less than min or greater than max.

You can assume the input list is not null, and is comprised of `SListNode`'s

You can also assume the `head` attribute in `SList`, and the `next` and `value` attributes in `SListNode`

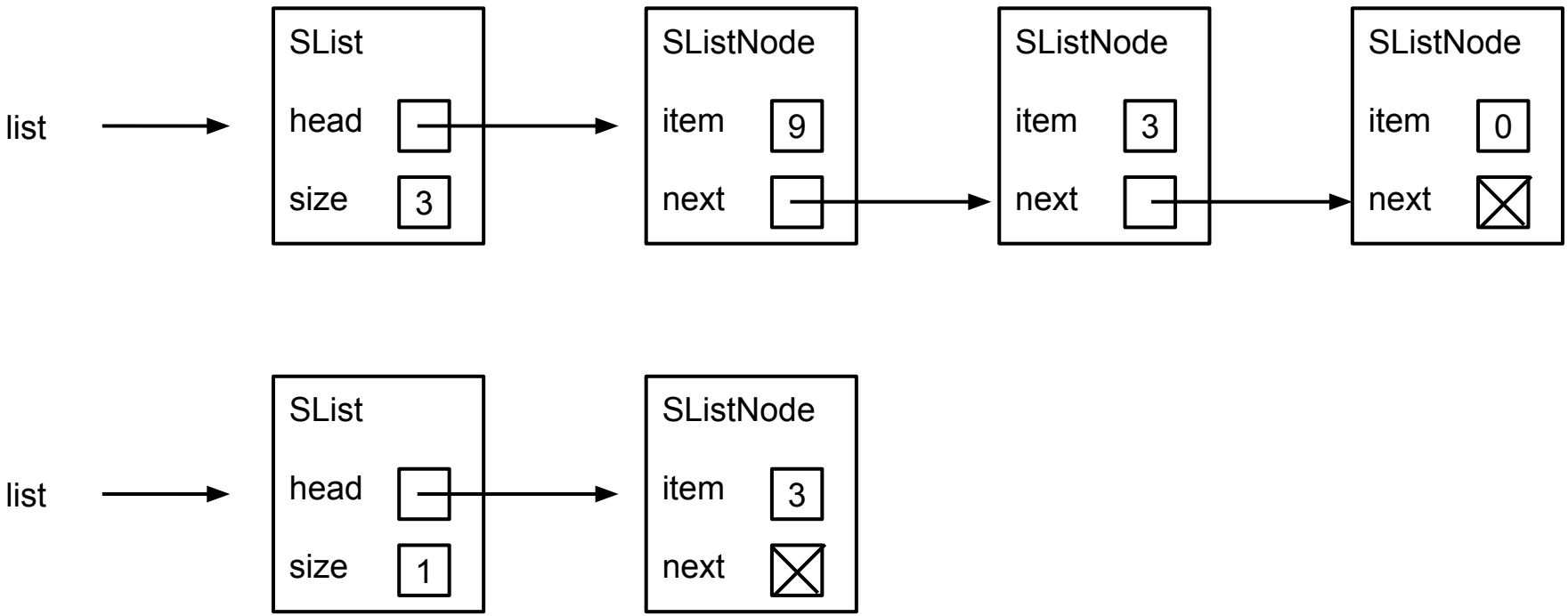
Singly Linked Lists!

```
trim(3, 5, list);
```



Singly Linked Lists!

```
trim(3, 5, list);
```



Singly Linked Lists!

```
public void trim(int min, int max, SList list) {
  SListNode curr = list.head.next;
  SListNode prev = list.head;

}
```

Singly Linked Lists!

```
public void trim(int min, int max, SList list) {
  SListNode curr = list.head.next;
  SListNode prev = list.head;
  while (curr != null) {
    if ((curr.value < min) || (curr.value > max)) {
      prev.next = curr.next;
      curr = curr.next;
    }
  }

}
```

Singly Linked Lists!

```
public void trim(int min, int max, SList list) {
  SListNode curr = list.head.next;
  SListNode prev = list.head;
  while (curr != null) {
    if ((curr.value < min) || (curr.value > max)) {
      prev.next = curr.next;
      curr = curr.next;
    } else {
      prev = curr;
      curr = curr.next;
    }
  }

}
```

Singly Linked Lists!

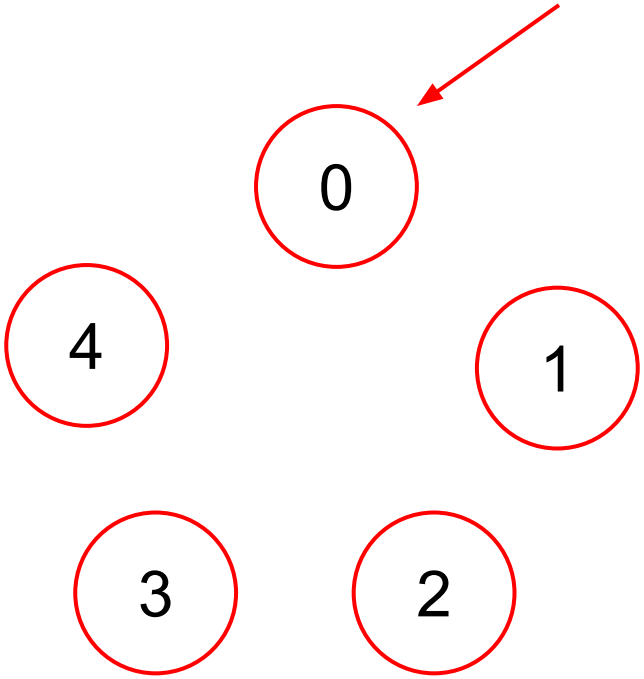
```
public void trim(int min, int max, SList list) {
  SListNode curr = list.head.next;
  SListNode prev = list.head;
  while (curr != null) {
    if ((curr.value < min) || (curr.value > max)) {
      prev.next = curr.next;
      curr = curr.next;
    } else {
      prev = curr;
      curr = curr.next;
    }
  }
  if ((list.head.value < min) || (list.head.value > max)) {
    list.head = list.head.next;
  }
}
```

Try drawing the pointers!

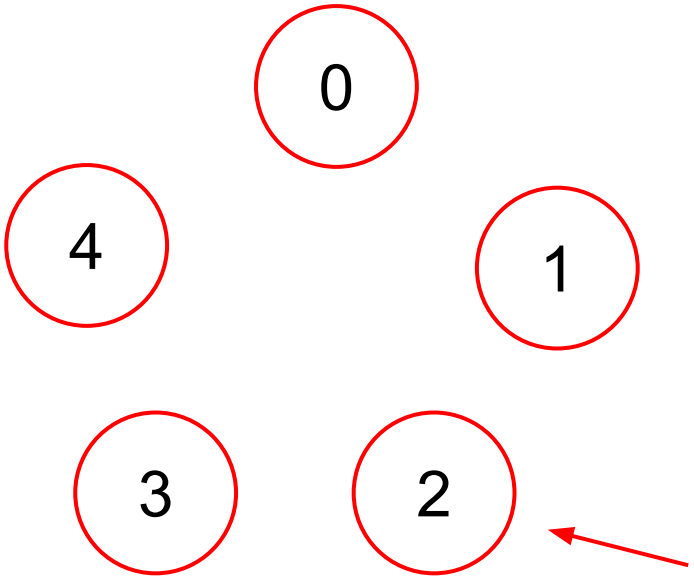
Confuzzling

You have **n** people sitting around in a circle, conveniently labelled **0, 1, …, n-1**. You start counting people off, starting at **0**. Then you count **k** people, and remove the **k**th person from the circle. Continue counting people off, and removing, until you have **1** person left. Who is that person?

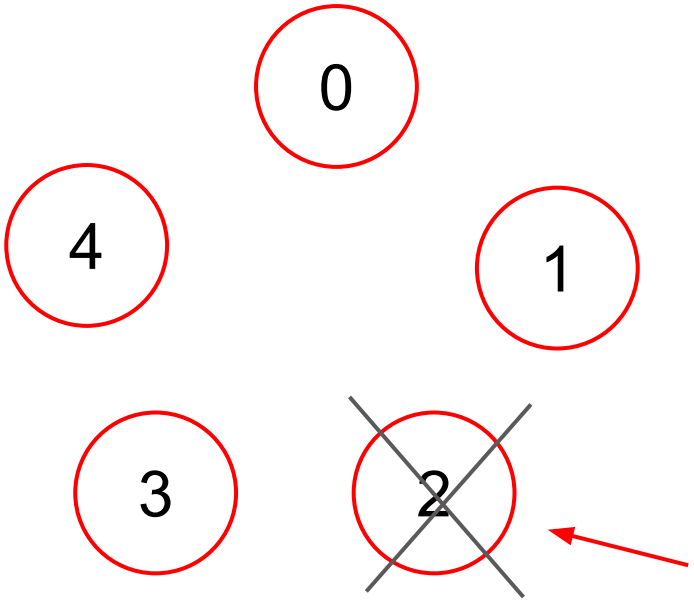
Example, n = 5, k = 2



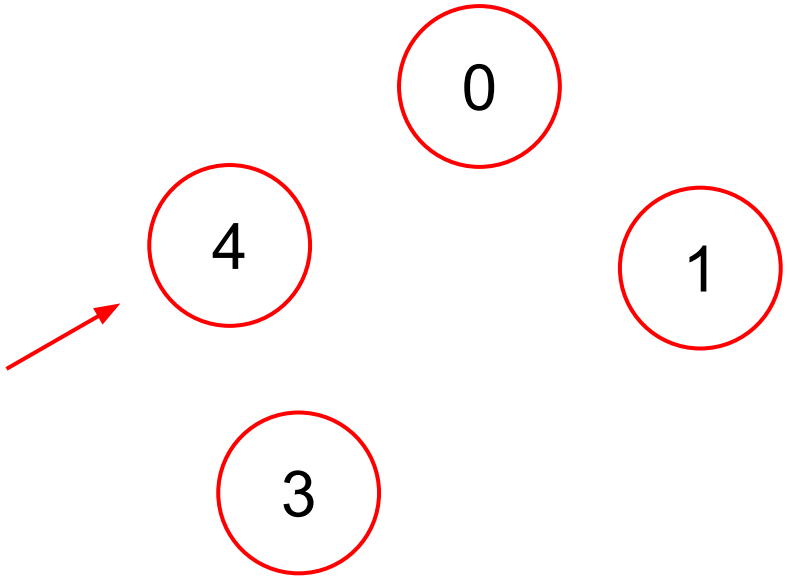
Example, n = 5, k = 2



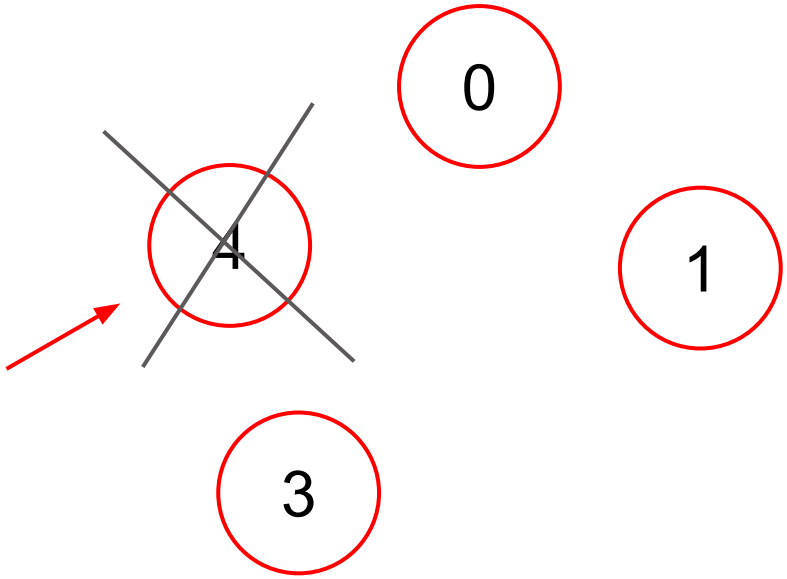
Example, n = 5, k = 2



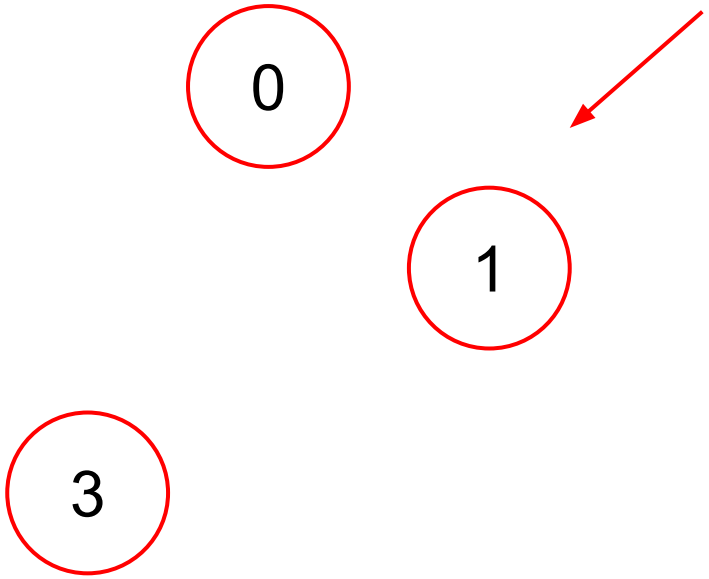
Example, n = 5, k = 2



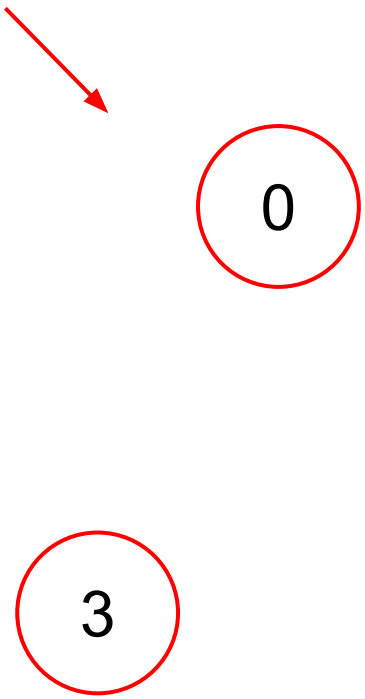
Example, n = 5, k = 2



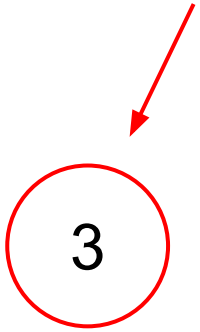
Example, n = 5, k = 2



Example, n = 5, k = 2



Example, n = 5, k = 2



Return 3!

Confuzzling

You have **n** people sitting around in a circle, conveniently labelled **0, 1, …, n-1**. You start counting people off, starting at **0**. Then you count **k** people, and remove the **k**th person from the circle. Continue counting people off, and removing, until you have **1** person left. Who is that person?

```
public int confuzzle(int n, int k) {
  /* YOUR CODE HERE */
}
```

Wait,

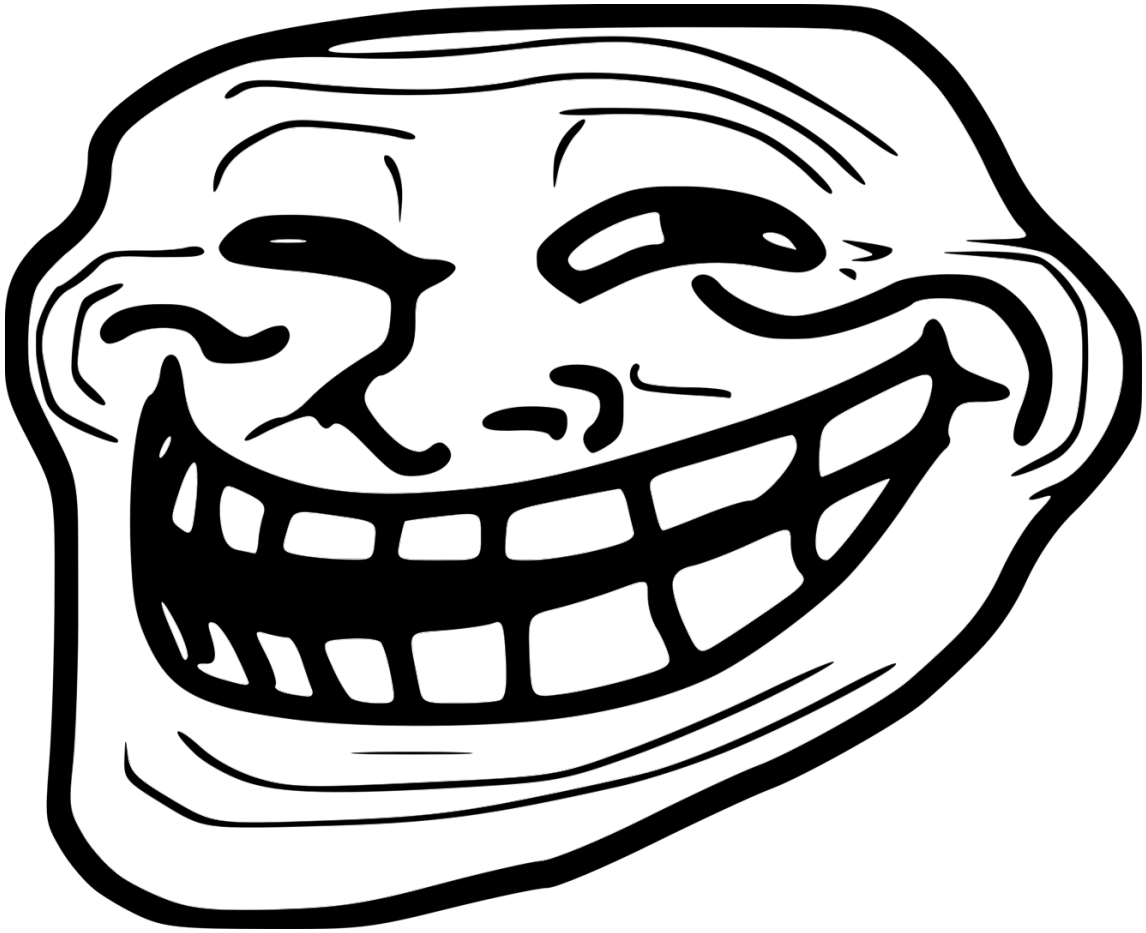
Wait, isn’t this…

Wait, isn’t this… DLIST - Sentinel ?!?!

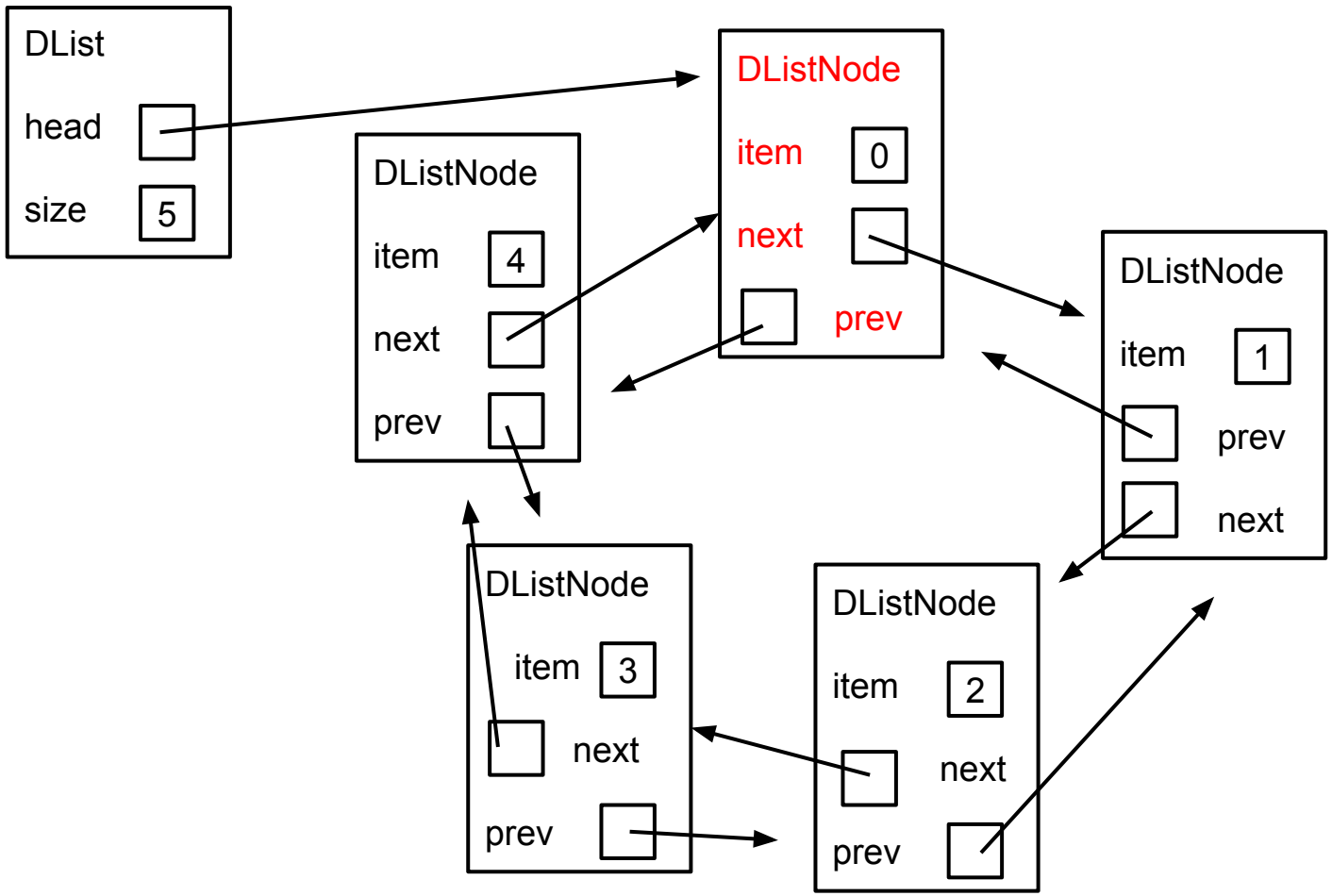
Wait, isn’t this… DLIST - Sentinel ?!?!

YEP

Wait, isn’t this… DLIST - Sentinel ?!?!



Wait, isn't this… DLIST - Sentinel ?!?!



Use DLists!

```
public class DList {
    DListNode head; //sentinel
    int size;
    public DList();
    public void insertFront(int i);
    public void insertEnd(int i);
    public void deleteNode(DListNode n);
}
```

*Assume that deleteNode can handle deletion of sentinel
*Assume that deleteNode does not remove the prev,next pointers of the deleted node.

Use DLists!

```
public int confuzzling(int n, int k){
    DList list = new DList();
    list.head.item = 0;
    for(int i = 1; i < n; i++)
        list.insertEnd(i);

}
```

Use DLists!

```
public int confuzzling(int n, int k){
    DList list = new DList();
    list.head.item = 0;
    for(int i = 1; i < n; i++)
        list.insertEnd(i);

    DListNode pointer = list.head;
    while(list.size > 1) {
        for(int steps = 0; steps < k; steps++)
            pointer = pointer.next;
        list.remove(pointer);
    }
    return pointer.item;
}
```

Inheritance

Basics:

- Natural class hierarchy!
- Subclasses **extend** Superclasses

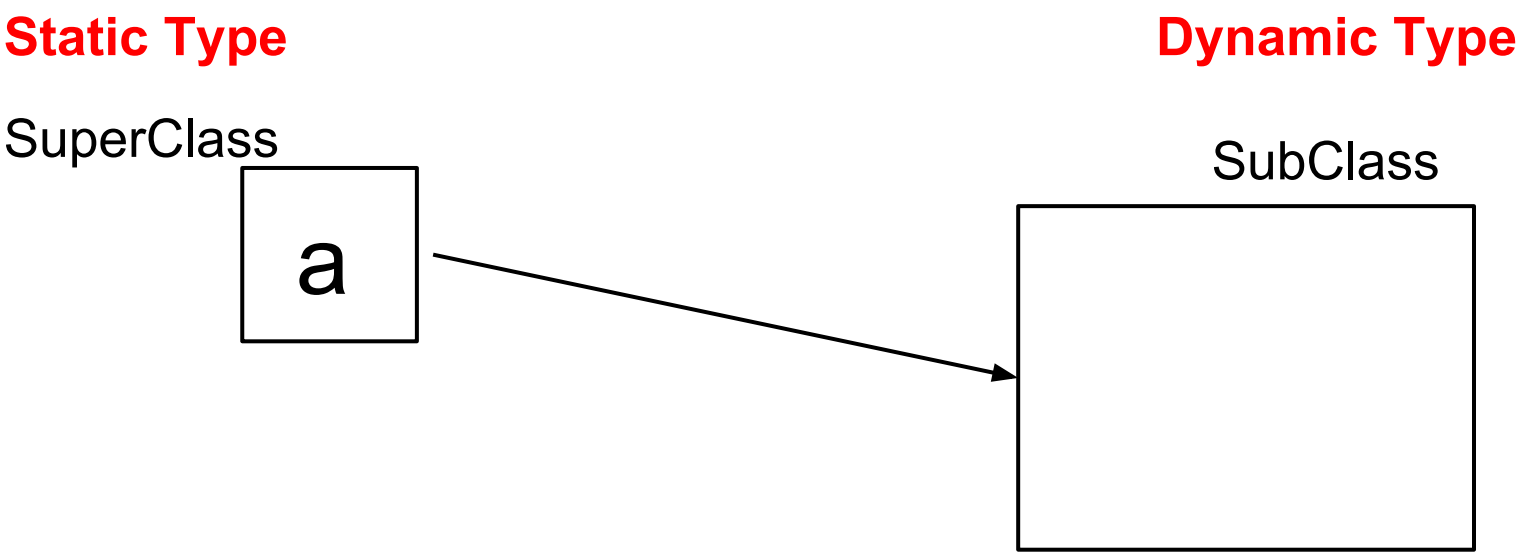
```
SuperClass a = new SubClass();
```

Static Type

Dynamic Type

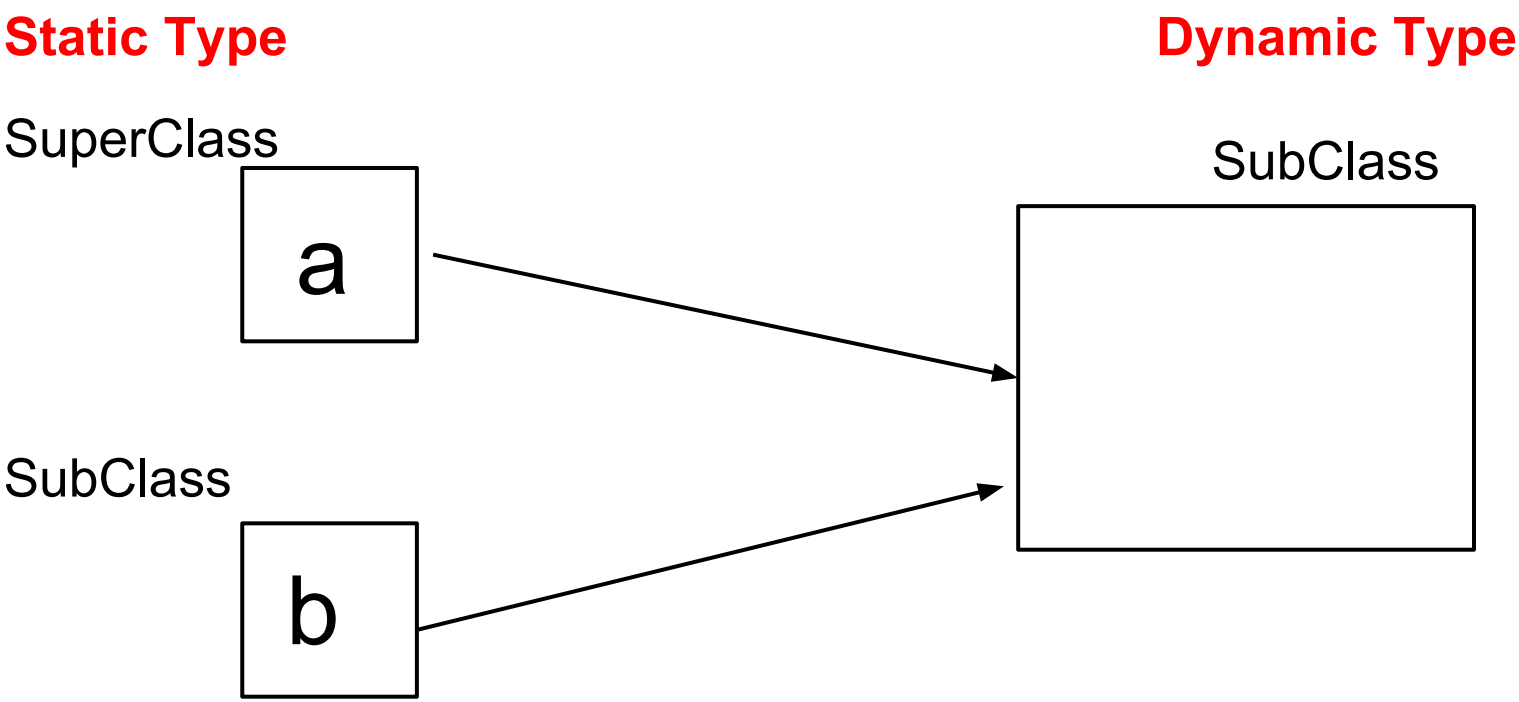
Inheritance

```
SuperClass a = new SubClass();
```



Inheritance

```
SuperClass a = new SubClass();
SubClass b = a;
```



Inheritance

Constructors:

There is **always** an implied call to the superclass constructor on the **FIRST LINE.**

```
class Child extends Parent {
    public Child() {
        System.out.println("hi!");
    }
}
```

```
class Child extends Parent {
    public Child() {
        super();
        System.out.println("hi!");
    }
}
```

these are the same!

(side note: the explicit call to super() can only be in the first line)

Inheritance

Dynamic Method Lookup:

Suppose we have the code cat.eat(). How do we determine what this does?

1. Does the **static type** of cat have the method eat()? If no, compiler error.
2. If yes, check the **dynamic type** of cat to see if eat() is **overridden**.

   If overridden, run the **dynamic class's method**. If not overridden, run the **static class's method**.

overridden methods must have the same **signature** (method name, argument types)

Field Shadowing:

We always consider the **static type** for looking up attributes

(for example, cat.name will look at the name in cat's static type)

Inheritance

Polymorphism:

1. If the static type is a subclass of the dynamic type, COMPILE-TIME ERROR

   ex: Cat c = new Animal();

2. We can **cast** "up" to any superclass without any problems

   ex: Cat c = new Cat();

   ((Animal) c) is a valid cast.

3. We can only cast "down" so far as the object's dynamic type

   --> if we cast "down" to a subclass below the dynamic type, RUN-TIME ERROR

   ex: Animal a = new Cat();

   ((Cat) a) is a **valid** cast.

   ex: Animal a = new Animal();

   ((Cat) a) will give us a **run-time error**!!

(inspiration from Summer '12, MT1)

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Superhero();
superhero.punch(superhero);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Superhero();
superhero.punch(superhero);
```

"I'M A SUPERHERO"
"BOOM I'M A SUPERHERO"

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman("I'M
BATMAN!");
batman.punch(batman);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman("I'M BATMAN!");
batman.punch(batman);
```

"I'M A SUPERHERO"
"I'M BATMAN!"
"wat."

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Superhero();
batman.punch(batman);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Superhero();
batman.punch(batman);
```

COMPILE-TIME ERROR!

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Batman();
superhero.punch( (Batman) superhero);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Batman();
superhero.punch( (Batman) superhero);
```

"I'M A SUPERHERO"
"BOOM I'M A SUPERHERO"
"BOOM BATMAN!"

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman();
((Superhero) batman).punch(batman);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman();
((Superhero) batman).punch(batman);
```

"I'M A SUPERHERO"
"BOOM I'M A SUPERHERO"
"BOOM BATMAN!"

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Superhero();
superhero.punch( (Batman) superhero);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Superhero();
superhero.punch( (Batman) superhero);
```

RUN-TIME ERROR!

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Batman();
superhero.punch( (Batman) superhero);
```

Inheritance

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }

    public void punch(Superhero a) {
        System.out.println("BOOM " + s);
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Superhero superhero = new Batman();
superhero.punch( (Batman) superhero);
```

"I'M A SUPERHERO"
"BOOM I'M A SUPERHERO"
"BOOM BATMAN!"

Inheritance

NOTE the changed source code!

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman();
((Superhero) batman).punch(batman);
```

Inheritance

NOTE the changed source code!

```java
public class Superhero {
    String s;
    public Superhero() {
        s = "I'M A SUPERHERO";
        System.out.println(s);
    }

    public void punch() {
        System.out.println("Punch! Punch!");
    }
}
```

```java
public class Batman extends Superhero {
    String s;
    public Batman() {
        s = "NANANANANANANA";
    }

    public Batman(String s) {
        this.s = s;
        System.out.println(this.s);
    }

    public void punch(Superhero v) {
        s = "BATMAN!";
        super.punch(v);
        System.out.println("BOOM " + s);
    }

    public void punch(Batman b) {
        System.out.println("Wat.");
    }
}
```

```java
Batman batman = new Batman();
((Superhero) batman).punch(batman);
```

COMPILE-TIME ERROR

Good luck on your midterm! ☺

HKN Office Hours

Monday - Friday, 11:00am-5:00pm

345 Soda

290 Cory

hkn.eecs.berkeley.edu