Announcement

Final exam: Friday, December 10 at 14:00 at the GYM

Recursion and linked-lists

```
public class Node {
  private Object data;
  private Node   next;
  public Node(Object d, Node n)
  {
    data = d;
    next = n;
  }
  public Object get_data() { return data; }
  public Node   get_next() { return next; }
  public void set_data(Object d)
  {
    data = d;
  }
  public void set_next(Node n)
  {
    next = n;
  }
}
```

Recursion and linked-lists

```
public class LinkedList
{
  private Node first;
  public LinkedList() { first = null; }
  public int length()
  {
    int counter = 0;
    Node p = first;
    while (p != null) {
      p = p.get_next();
      i++;
    }
    return counter;
  }
}
```

Recursion and linked-lists

- What is a list of objects?
  - The empty list $\langle\rangle$ is a list.
  - If l is some list, then $\langle x, l\rangle$ is a list

- For example:
  - $l_1 = \langle 3, \langle\rangle\rangle$ is a list
  - $l_2 = \langle 5, l_1\rangle = \langle 5, \langle 3, \langle\rangle\rangle\rangle$ is a list
  - $l_3 = \langle -6, l_2\rangle = \langle -6, \langle 5, \langle 3, \langle\rangle\rangle\rangle\rangle$ is a list

Recursion and linked-lists

```
public abstract class List
{
}
public class EmptyList extends List
{
}
public class Cons extends List
{
  private Object head;
  private List   tail;
  public Cons(Object h, List t)
  {
    head = h;
    tail = t;
  }
  public Object get_head() { return head; }
  public List   get_tail() { return tail; }
  // ...
}
```

Recursion and linked-lists

```
public class Test {
  public static void main(String[] args)
  {
    String a = ‘‘hello’’, b = ‘‘bonjour’’, c = ‘‘hola’’;
    List l0 = new EmptyList();
    List l1 = new Cons(b, l0);
    List l2 = new Cons(a, l1);
    List l3 = new Cons(c, l2);
  }
}
```

Recursion and linked-lists

```
public class Test {
  public static void main(String[] args)
  {
    String a = “hello”, b = “bonjour”, c = “hola”;
    List l3 = new Cons(c,
                       new Cons(a,
                                new Cons(b,
                                         new EmptyList())));
  }
}
```

Recursion and linked-lists

The length of a list:

$$length(l) = \begin{cases} 0 & \text{if } l = \langle\rangle \\ 1 + length(t) & \text{if } l = \langle x, t\rangle \end{cases}$$

Recursion and linked-lists

```
public class ListOperations
{
  public static int length(List s)
  {
    if (s instanceof EmptyList) {
      return 0;
    }
    Cons pair = (Cons)s;
    return 1 + length(pair.get_tail());
  }
}
```

Recursion and linked-lists

```
public class Test {
  public static void main(String[] args)
  {
    String a = ‘‘hello’’, b = ‘‘bonjour’’, c = ‘‘hola’’;
    List l0 = new EmptyList();
    List l1 = new Cons(b, l0);
    List l2 = new Cons(a, l1);
    List l3 = new Cons(c, l2);
    int n;
    n = ListOperations.length(l3);
  }
}
```

Recursion and linked-lists

Finding out if an element is in a list:

$$member(x,l) = \begin{cases} false & \text{if } l = \langle\rangle \\ true & \text{if } l = \langle x,t\rangle \\ member(x,t) & \text{if } l = \langle y,t\rangle \text{ and } x \neq y \end{cases}$$

Recursion and linked-lists

```
public class ListOperations
{
  // ...
  public static int member(Object x, List s)
  {
    if (s instanceof EmptyList) {
      return false;
    }
    Cons pair = (Cons)s;
    if (x.equals(pair.get_head())) {
      return true;
    }
    return member(x, pair.get_tail());
  }
}
```

Recursion and linked-lists

```
public class Test {
  public static void main(String[] args)
  {
    String a = “hello”, b = “bonjour”, c = “hola”;
    List l0 = new EmptyList();
    List l1 = new Cons(b, l0);
    List l2 = new Cons(a, l1);
    List l3 = new Cons(c, l2);
    boolean b;
    b = ListOperations.member(“hiya”, l3);
  }
}
```

Applications (Simulation)

```
class Customer { ... }
class SuperMarket {
  Queue line;
  SuperMarket() { line = new Queue(); }
  void process(Customer c) { ... }
  void run()
  {
    while (true) {
      int coin = (int)(Math.random() * 2);
      if (coin == 1) {
        Customer first = line.peek();
        process(first);
        line.dequeue();
      }
      else {
        line.enqueue(new Customer());
      }
    }
  }
}
```

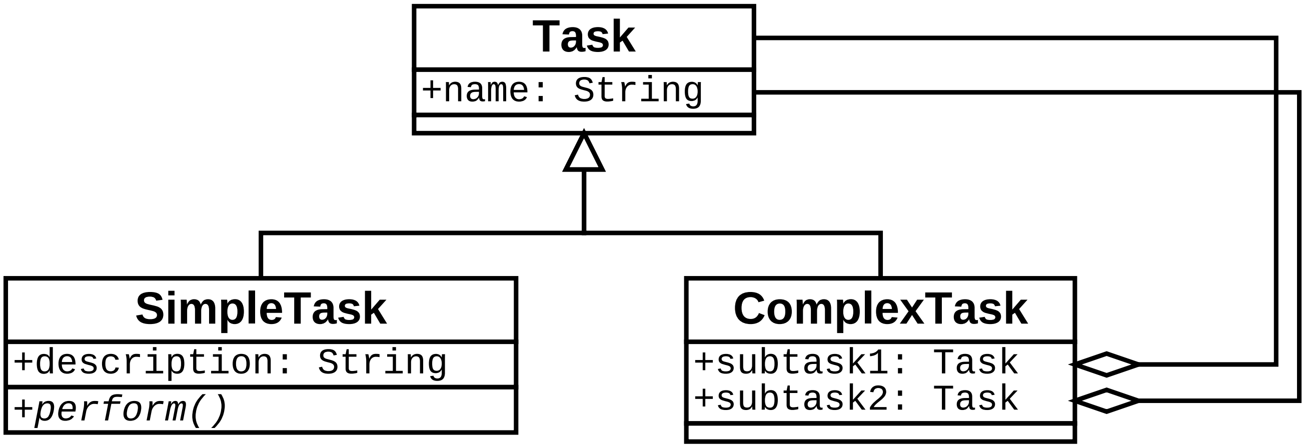
Applications (reverse)

```
static String reverse(String s)
{
  String r = "";
  Stack stack = new Stack();
  int i = 0;
  while (i < s.length()) {
    stack.push(new Character(s.charAt(i)));
    i++;
  }
  while (!stack.isempty()) {
    Character c = (Character)stack.top();
    r = r + c.charValue();
    stack.pop();
  }
  return r;
}
```

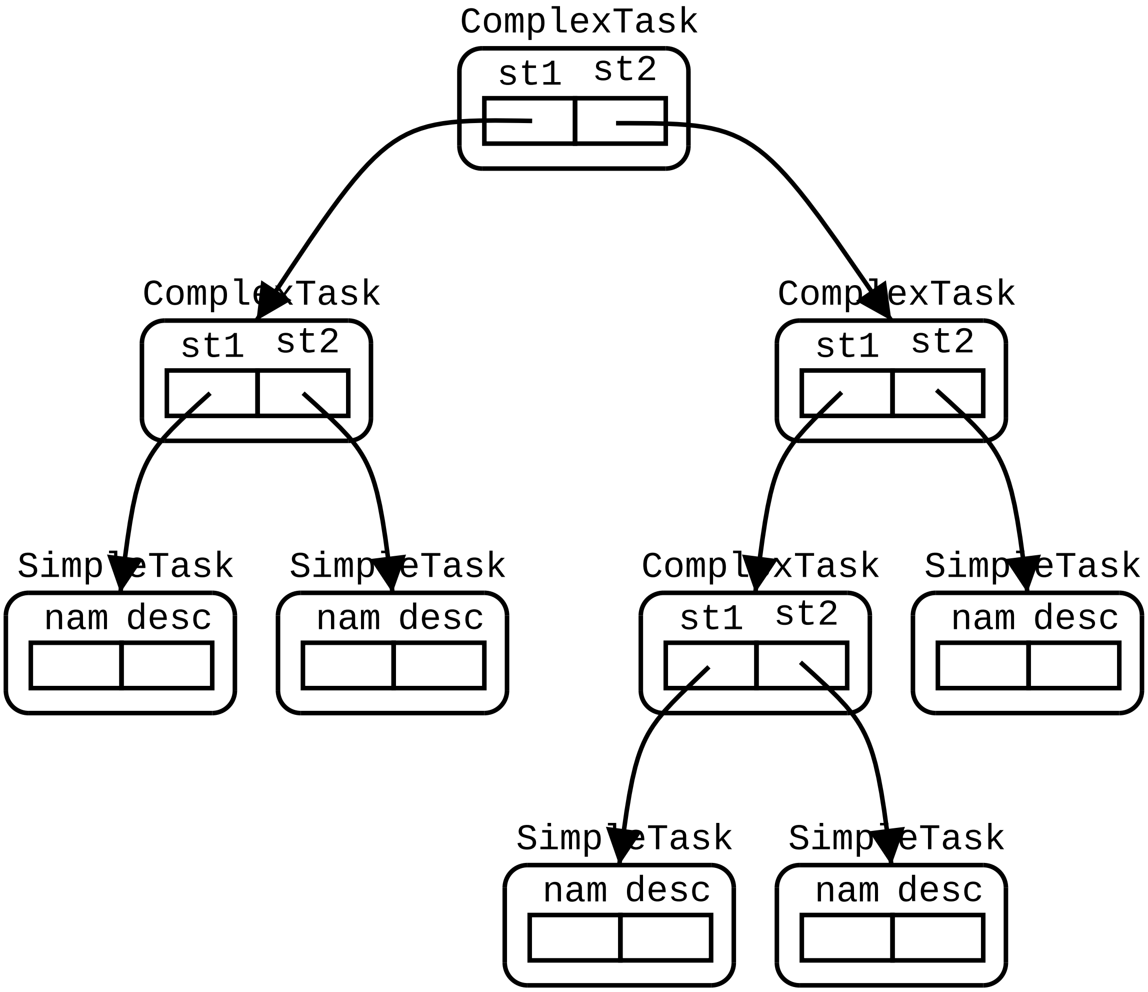
Trees

- A tree is a non-linear data-structure
- If there is an arrow from a node x to a node y, we call x the *parent* of y and y *a child* of x.
- If there is a *path from* x to z, we say that x is an ancestor of z and z is a descendant of x.
- A *leave* is a node with no children
- A data-structure is a tree if:
  - Every node has only one parent
  - There is no node with an arrow to any of its ancestors (there are no "loops")
  - There is exactly one node with no parent (the "root")
- Normal trees are finite: no infinite paths.
- A tree is *binary* if every node has two children
- A unary tree is a list

Binary Trees



Binary Trees



Binary Trees

```
abstract class Task {
  String name;
}

class SimpleTask extends Task {
  String description;
  void perform()
  {
    System.out.println(name+“:”+description);
    //...
  }
}

class ComplexTask extends Task {
  Task subtask1, subtask2;
}
```

Binary Trees

- Processing trees using recursion

```
class Worker {
  void work(Task t)
  {
    if (t instanceof SimpleTask) {
      ((SimpleTask)t).perform();
    }
    else if (t instanceof ComplexTask) {
      work(((ComplexTask)t).subtask1);
      work(((ComplexTask)t).subtask2);
    }
  }
}
```

Binary Trees

- Processing trees using stacks

```
class Worker {
  void work(Task t)
  {
    Stack s = new Stack();
    s.push(t);
    while (!s.isempty()) {
      Task temp = s.top();
      s.pop();
      if (temp instanceof SimpleTask) {
        ((SimpleTask)t).perform();
      }
      else {
        s.push(((ComplexTask)temp).subtask2);
        s.push(((ComplexTask)temp).subtask1);
      }
    }
  }
}
```

Data structures zoo

- Other data-structures: sets, bags, priority queues, heaps, binary trees, n-ary trees, red-black trees, AVL trees, graphs, hyper-graphs, hi-graphs, dictionaries/mappings, etc.

- The selection of data-structure has a major impact on the efficiency of an algorithm.

What this course is about

- This course is an introduction to *computer programming*
- Computer programming: solving problems involving information by means of a computer

What this course is *not* about

- This course is ***not*** about...
  - ...how to use a computer
  - ...how to use software applications
  - ...how to use the Operating System
  - ...how to send e-mail
  - ...how to surf the Web
  - ...how to create Web pages
  - ...how to fix your printer
  - ...how to become a hacker
  - ...how to manage a computer system (installing software, fixing problems, etc.)
- There is no course in Computer Science about how to use computers, in the same way that there is no course in Mechanical Engineering that teaches how to drive a car or operate some machinery.

Objectives

- To learn:
  - ...a methodology to understand and solve problems involving information
  - ...how to think computationally
  - ...how to create simple algorithms
  - ...how to design and implement computer programs using the Java programming language
  - ...how to solve problems in an Object-Oriented manner
- **This is neither a "computers course" nor a "Java course."**

The end