

Introduction to Computer Science II

CS112-2012S-25

Final Review II

David Galles

Department of Computer Science
University of San Francisco

25-0: **Review**

- Linked Lists
- Inheritance (including polymorphism)
- Potpourri (Exceptions, etc)

25-1: Linked Lists

```
public class LinkedListNode
{
    private Object data;
    private LinkedListNode next;
    public LinkedListNode(Object data)
    {
        this.data = data;
        next = null;
    }
    public LinkedListNode(Object data, LinkedListNode next)
    {
        this.data = data;
        this.next = next;
    }
    public Object getData()      public Object setData(object data)
    {
        return data;            this.data = data;
    }
    // More on next slide
}
```

25-2: Linked Lists

```
public class LinkedListNode
{
    private Object data;
    private LinkedListNode next;

    // Rest of class on previous slide

    public Object getNext()
    {
        return next;
    }

    public Object setNext(LinkedListNode next)
    {
        this.next = next;
    }
}
```

25-3: Linked Lists

- Traversing a linked list

```
LinkedListNode tmp = lst;
while (tmp != null)
{
    // Do something with tmp
    tmp = tmp.getNext();
}
```

25-4: **Linked Lists**

- Write a function that prints out a linked list
- `void print(LinkedListNode list)`

25-5: Linked Lists

```
void print(LinkedListNode list)
{
    LinkedListNode tmp = list
    while (tmp != null)
    {
        System.out.println(tmp.getData());
        tmp = tmp.getNext();
    }
}
```

- How would we do this recursively?

25-6: **Linked Lists**

- Recursive function:
 - What is an easy version of the problem that we could solve immediately?
 - How can we make the problem smaller?
 - Assuming that we could solve the smaller problem, how could we solve the original larger problem?

25-7: Linked Lists

- What is an easy version of the problem?
 - Empty list
- How can we make the problem smaller?
 - `list.getNext()` is a smaller list
- Assuming that we could solve the smaller problem, how could we solve the original larger problem?
 - Print first element of list, then print smaller list

25-8: Linked Lists

```
void print(LinkedListNode list)
{
    if (list != null)
    {
        System.out.println(list.getData());
        print(list.getNext());
    }
}
```

- What if we wanted to print out the list *in reverse*?

25-9: **Linked Lists**

- What is an easy version of the problem?
 - Empty list
- How can we make the problem smaller?
 - `list.getNext()` is a smaller list
- Assuming that we could solve the smaller problem, how could we solve the original larger problem?
 - Print smaller list reversed, then print first element

25-10: Linked Lists

```
void printReverse(LinkedListNode list)
{
    if (list != null)
    {
        printReverse(list.getNext());
        System.out.println(list.getData());
    }
}
```

- How hard would this be to do iteratively?

25-11: Linked Lists

```
public void printReversed(LinkedListNode list)
{
    StringLinkedListNode last = list;
    StringLinkedListNode prev = null;

    while (prev != head)
    {
        prev = head;
        while (prev.next != last)
        {
            prev = prev.next;
        }
        System.out.println(prev.data);
        last = prev;
    }
}
```

25-12: **Linked Lists**

- From Monday's FinalReview: largestClump for linked list

25-13: Linked Lists

```
public void largestClump(LinkedListNode list)
{
    int currentClump = 1;    int largestClump = 1;
    while (list.next != null)
    {
        if (list.getData().equals(list.getNext().getData()))
        {
            currentClump++;
            if (currentClump > largestClump)
            {
                largestClump = currentClump;
            }
        }
        else
        {
            currentClump = 1;
        }
    }
    return largestClump;
}
```

25-14: Inheritance

- Add “extends <classname>” to class definition
 - `class Apartment extends Building { ... }`
- Defines an “is-a” relationship
 - Apartment is a building
- Defines a superclass/subclass relationship
 - Building is the superclass
 - Apartment is the subclass

25-15: Inheritance

- Add “extends <classname>” to class definition
 - `class Apartment extends Building { ... }`
- Subclass inherits all of the methods / data from the superclass.
 - Examples from code

25-16: Polymorphism

- We have an array of Objects ...
- What can we do with it?
 - Print out the object
 - Convert it to string (using toString)

How can we do something more useful?

25-17: Polymorphism

- We can “override” methods described in a superclass
 - Create a method in the subclass with the same “signature” as the superclass
 - Same name, same number and type of parameters
 - Subclass will use the new definition of the method

25-18: Polymorphism

```
class A
{
    void print()
    {
        System.out.println("Hello from A");
    }
}
class B extends A
{
    void print()
    {
        System.out.println("Hello from B");
    }
}
```

```
A classA = new A();
B classB = new B();
```

```
classA.print();
classB.print();
```

25-19: Polymorphism

```
class A
{
    void print()
    {
        System.out.println("Hellow from A");
    }
}
class B extends A
{
    void print()
    {
        System.out.println("Hellow from B");
    }
}

A classA = new B();

classA.print();
```

25-20: Polymorphism

- Superclass contains a method “Print”
- Subclass overrides the method “Print”
- Assign a subclass value to a superclass variable
- Call the print method of the superclass variable
 - Uses the subclass version

25-21: Polymorphism

- We've actually seen this before
 - `toString()`

25-22: Polymorphism

```
class A
{
    void print()
    {
        System.out.println("Hello from A");
    }
}

class B extends A
{
    void print()
    {
        System.out.println("Hello from B");
    }
}
```

```
In main:
-----

A aArray = new A[5];
aArray[0] = new A();
aArray[1] = new A();
aArray[2] = new B();
aArray[3] = new B();
aArray[4] = new A();

for (int i = 0; i < 5; i++)
{
    aArray[i].print();
}
```


25-23: Polymorphism

```
class A {
    void print() {
        System.out.println("Hello from A")
    }
}
class B extends A {
    void print() {
        System.out.println("Hello from B")
    }
}
class C extends A {
    void print() {
        System.out.println("Hello from C")
    }
}
class D extends B {
    void print() {
        System.out.println("Hello from D")
    }
}
```

In main:

```
A aArray = new A[5];
aArray[0] = new B();
aArray[1] = new C();
aArray[2] = new D();
aArray[3] = new C();
aArray[4] = new D();

for (int i = 0; i < 5; i++)
{
    aArray[i].print();
}
```

25-24: Exceptions

```
class Silly {
    int badFunc()
    {
        System.out.println("A");
        int y = x / 0;
        System.out.println("B");
    }
    void foo()
    {
        System.out.println("C");
        badFunc();
        System.out.println("D");
    }

    void bar()
    {
        System.out.println("E");
        foo();
        System.out.println("F");
    }
    void start()
    {
        try
        {
            System.out.println("G");
            bar();
            System.out.println("H");
        }
        catch (Exception e)
        {
            System.out.println("I");
        }
        System.out.println("J");
    }
}
```

25-25: Exceptions

```
class Silly {
    int badFunc()
    {
        System.out.println("A");
        int y = x / 2;
        System.out.println("B");
    }
    void foo()
    {
        System.out.println("C");
        badFunc();
        System.out.println("D");
    }

    void bar()
    {
        System.out.println("E");
        foo();
        System.out.println("F");
    }
    void start()
    {
        try
        {
            System.out.println("G");
            bar();
            System.out.println("H");
        }
        catch (Exception e)
        {
            System.out.println("I");
        }
        System.out.println("J");
    }
}
```