Tentamen i Objektorienterad modellering och design

Lösningar

1.
```
public abstract class Instruction {
    protected List<Address> addressList;
    public abstract void execute(Memory memory);
}
public class Copy extends Instruction {
    public void execute(Memory memory) {
        memory.copy(addressList.get(0), addressList.get(1));
    }
}
public class Add extends Instruction {
    public void execute(Memory memory) {
        memory.add(addressList.get(0), addressList.get(1), addressList.get(2));
    }
}
```

2.
```
public abstract class Instruction {
    protected List<Address> addressList;
    public abstract String opString();

    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append(opString());
        for (Address address : addressList) {
            builder.append(' ').append(address);
        }
        return builder.toString();
    }
}
public class Copy extends Instruction {
    public String opString() {
        return "COPY";
    }
}
public class Add extends Instruction {
    public String opString() {
        return "ADD";
    }
}
```

3.
```
public interface StringComparator {
    public int compareTo(String string1, String string2);
}
public class StandardComparator implements StringComparator {
    public int compareTo(String string1, String string2) {
        return string1.compareTo(string2);
    }
```

```
}
public class StringList extends ArrayList<String> {
    private StringComparator comparator = new StandardComparator();

    public void setComparator(StringComparator comparator) {
        this.comparator = comparator;
    }

    public void sort() {
        for (int i = 0; i < size(); i++) {
            for (int j = i + 1; j < size(); j++) {
                if (comparator.compareTo(get(i), get(j)) > 0) {
                    // omissions
                } else {
                    // omissions
                }
            }
        }
    }
}
```

4. I uppgiften finns tillfälle att tillämpa alla principerna och många mönster. För full poäng krävs att lösningen har god kvalitet enligt de kriterier som tillämpats i kursen.

```
public abstract class Unit {
    private Extension extension;
    protected String name;

    protected Unit(String name, Extension extension) {
        this.name = name;
        this.extension = extension;
    }

    public String toString() {
        return name + extension;
    }

    public static void main(String[] args) {
        List<Interface> list = new ArrayList<Interface>();
        list.add(new Interface("C"));
        list.add(new Interface("D"));
        Unit unit = new Class("A", new WithExtension(new Class("B")), list);
        System.out.println(unit);
    }
}

public class Class extends Unit {
    private List<Interface> interfaces;

    public Class(String name, Extension extension, List<Interface> interfaces) {
        super(name, extension);
        this.interfaces = interfaces;
```

```
    }

    public String toString() {
        StringBuilder builder = new StringBuilder("class ");
        builder.append(super.toString());
        if (interfaces.size() > 0) {
            builder.append(" implements ");
            for (Interface interface_ : interfaces) {
                builder.append(interface_.name);
                builder.append(", ");
            }
            int length = builder.length();
            builder.delete(length - 2, length);
        }
        return builder.toString();
    }
}

public class Interface extends Unit {
    public Interface(String name, Extension extension) {
        super(name, extension);
    }

    public String toString() {
        return "interface " + super.toString();
    }
}

public interface Extension {
}

public class NoExtension implements Extension {
    public String toString() {
        return "";
    }
}

public class WithExtension implements Extension {
    private Unit unit;

    public WithExtension(Unit unit) {
        this.unit = unit;
    }

    public String toString() {
        return " extends " + unit.name;
    }
}
```

5. Lägg till klassen

```
public class ErrorExpr implements Expr {
```

```
    public double value(Sheet sheet) {
        throw new RuntimeException("Circular dependence");
    }
}
```

och modifiera `Sheet.put`

```
public class Sheet {
    private HashMap<Address, Expr> map;
    public void put(Address address, Expr expr) {
        map.put(address, new ErrorExpr());
        expr.value(this);
        map.put(address, expr);
    }
}
```