Tentamen i Objektorienterad modellering och design

Lösningar

1.
```
public interface Log {
  public void add(Instruction instruction);
}
public class NoLog implements Log {
  public void add(Instruction instruction) {}
}
public class ListLog implements Log {
  private List<Instruction> list = new ArrayList<Instruction>();
  public void add(Instruction instruction) {
    list.add(instruction);
  }
}
public class Computer {
    private Memory memory;
    private Program program;
    private ProgramCounter programCounter;
    private Log log = new NoLog();

    public setLog(Log log) {
        this.log = log;
    }
    public void run() {
        programCounter.setCounter(0);
        while (programCounter.getCounter() >= 0) {
            Instruction instruction = program.get(programCounter.getCounter());
            log.add(instruction);
            programCounter.increment();
            instruction.execute(memory, programCounter);
        }
    }
}
```

2.
```
public class Memory {
  private Word memory[];
  public Word getWord(int address) {
    return memory[address];
  }
}

public class Add implements Instruction {
  private int  address1, address2, address3;
  public void execute(Memory memory, ProgramCounter programCounter) {
    Word word1 = memory.getWord(address1);
    Word word2 = memory.getWord(address2);
    Word word3 = memory.getWord(address3);
```

```
    word3.add(word1, word2);
  }
}
public interface Word {
  public void add(Word word1, Word word2);
}
public class ExtendedWord implements Word {
  private long high, low;
  public void add(Word word1, Word word2) {
  // omissions
  }
}
```

3.
```
protected abstract class ThreeOperandInstruction implements Instruction {
  private int address1, address2, address3;
  protected ThreeOperandInstruction(int address1, int address2, int address3) {
    this.address1 = address1;
    this.address2 = address2;
    this.address3 = address3;
  }
  abstract protected String opString();
  public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append(opString()). append(' ');
    builder.append(address1);
    builder.append(" ");
    builder.append(address2);
    builder.append(" ");
    builder.append(address3);
    builder.append(" ");
    return builder.toString();
  }
}
public class Add extends ThreeOperandInstruction {
  public Add(int address1, int address2, int address3) {
    super(address1, address2, address3);
  }
  protected String opString() {
    return "ADD";
  }
}
```

4.
```
public interface Expr {}
public class Not implements Expr {
  private Expr expr;
}
public class And implements Expr {
  private Expr expr1, expr2;
}
public class And implements Expr {
  private Expr expr1, expr2;
```

```
  }
```

5.
```
public class MemoryView extends JPanel implements Observer {
  private Memory memory;
  private JLabel label [];

  public MemoryView(Memory memory) {
    this.memory = memory;
    int size = memory.size();
    label = new JLabel[size];
    setLayout(new GridLayout(size / 8 + 1, 8, 2, 2));
    for (int i = 0; i < size; i++) {
      label[i] = new JLabel("        0");
      memory.getWord(i).addObserver(this);
      add(label[i]);
    }
  }
  public void update(Observable observable, Object object) {
    for (int i = 0; i < memory.size(); i++) {
      label[i].setText(memory.getWord(i)))
    }
  }
}
public abstract Word extends Observable {
  public abstract void add(Word word1, Word word2);
}
public class ExtendedWord extends Word {
  private long high, low;
  public void add(Word word1, Word word2) {
  // omissions
  setChanged();
  notifyObservers();
  }
}
```