Solutions to questions of lecture 2

version 1 - 18/02-04 KBG
version 2 - 23/09-04 KaØ

1a

First all fields are initialized to their default value (null, false or 0), then the initialization of the superclass' fields (in the order they appear in the code) and the execution of the constructor, then the subclass fields (in the order they appear) and then its constructor is executed.

forgetting the initial phase, the result is

```
AAA.a = "Hello"
AAA.b = null
AAA.x = 17
AAA.b = "G'day mate"
BBB.c = "Go'dag"
BBB.c = "davs"
```

1b
The conflicts arises since 'super(d)' is invoked before the initialization of 'd' is performed (this is performed after the initialization of the super class' fields and and execution the super class' constructor).

2
```
class Car {
        private Person owner;
        private String type;

        public Car(Person owner, String type){
                this.type = type;
                setOwner(owner);
        }

        public String toString(){
                return type;
        }
        public void setOwner(Person newOwner){
                if ( newOwner.car != null )
                        throw new Exception("You can only own one car");

                if (owner != null) // someone used to own this car
                        owner.setCar(null);
                owner = newOwner;
                newOwner.setCar(this);

        }

        public Person getOwner(){
                return owner;
        }

}
```

3
```
public Object clone(){
        try{

                Pen c = (Pen)super.clone();
                // since private fields are accessible from within other instances of the
same class,
                // we can alter the colour of the cloned pen
                if(col == Coler.RED) c.col = Color.BLUE;
                else if(col == Color.BLUE) c.col = Color.RED;
                // do nothing if the pen is any other color
                return c;
        }catch(CloneNotSupportedException e) {
                throw new RuntimeException("super class doesn’t implement cloneable");
        }
}
```

4

```
        /*      Exercise 4
         *      First remember to implement the interface Cloneable

                A simple solution is to make the cloned person NOT own a car.
                This is what is done below.
                An other solution is to say one cannot clone persons who owns a car.
                A third solution would be to provide the clone with a clone of the car.

         */
        public Object clone() {
                try {
                        Person p = (Person) super.clone();
                        p.car = null;
                        return p;
                }
                catch(CloneNotSupportedException e) {
                        throw new Error(e);
                }
        }

        // Clone test
        public static void main(String[] args)
        {
                Person p1 = new Person("Lars");
                Car c = new Car(p1,"Skoda");
                Person p2 = (Person) p1.clone();
                System.out.println( p1.getCar() );
                System.out.println( p2.getCar() );
                System.out.println( "and owners" );
                System.out.println( p1.getCar().getOwner() ); // Lars
                System.out.println( p2.getCar() ); // null
        }
```