Spencer Berg

Dr. Chetan Jaiswal

CS260 Object-Oriented Programming and Design

30 November 2016

Inheritance Homework

1. Rewriting with composition:

```
// BasePlusCommissionEmployee class now is uses composition with CommissionEmployee
// and accesses the class's private data via inherited public methods.

public class BasePlusCommissionEmployee // extends CommissionEmployee
{
	CommissionEmployee ce;
	private double baseSalary; // base salary per week

	// six-argument constructor
	public BasePlusCommissionEmployee(String first, String last, String ssn, double sales,
double rate, double salary) {
		// super( first, last, ssn, sales, rate );
		ce = new CommissionEmployee(first, last, ssn, sales, rate);
		setBaseSalary(salary); // validate and store base salary
	} // end six-argument BasePlusCommissionEmployee constructor

	// set base salary
	public void setBaseSalary(double salary) {
		if (salary >= 0.0)
			baseSalary = salary;
		else
			throw new IllegalArgumentException("Base salary must be >= 0.0");
	} // end method setBaseSalary

	// return base salary
	public double getBaseSalary() {
		return baseSalary;
	} // end method getBaseSalary

	// calculate earnings
	public double earnings() {
		return getBaseSalary() + ce.earnings();
	} // end method earnings
```

```
	// return String representation of BasePlusCommissionEmployee
	@Override // indicates that this method overrides a superclass method
	public String toString() {
		return String.format("%s %s\n%s: %.2f", "base-salaried", ce.toString(), "base
salary", getBaseSalary());
	} // end method toString

	public void setFirstName(String first) {
		ce.setFirstName(first); // should validate
	} // end method setFirstName

	// return first name
	public String getFirstName() {
		return ce.getFirstName();
	} // end method getFirstName

	// set last name
	public void setLastName(String last) {
		ce.setLastName(last);// should validate
	} // end method setLastName

	// return last name
	public String getLastName() {
		return ce.getLastName();
	} // end method getLastName

	// set social security number
	public void setSocialSecurityNumber(String ssn) {
		ce.setSocialSecurityNumber(ssn); // should validate
	} // end method setSocialSecurityNumber

	// return social security number
	public String getSocialSecurityNumber() {
		return ce.getSocialSecurityNumber();
	} // end method getSocialSecurityNumber

	// set gross sales amount
	public void setGrossSales(double sales) {
		if (sales >= 0.0)
			ce.setGrossSales(sales);
		else
			throw new IllegalArgumentException("Gross sales must be >= 0.0");
	} // end method setGrossSales
```

```
        // return gross sales amount
        public double getGrossSales() {
                return ce.getGrossSales();
        } // end method getGrossSales

        // set commission rate
        public void setCommissionRate(double rate) {
                if (rate > 0.0 && rate < 1.0)
                        ce.setCommissionRate(rate);
                else
                        throw new IllegalArgumentException("Commission rate must be > 0.0
 and < 1.0");
        } // end method setCommissionRate

        // return commission rate
        public double getCommissionRate() {
                return ce.getCommissionRate();
        } // end method getCommissionRate

        // calculate earnings

} // end class BasePlusCommissionEmployee
```

2. Lizard Class

```
public class Lizard extends Reptile
{
        private int lizardLength;
        private String lizardLocation;

        public Lizard(double brainSize, double eggSize, int lizardLength, String lizardLocation)
        {
                super(brainSize, eggSize);
                this.lizardLength = lizardLength;
                this.lizardLocation = lizardLocation;
        }

        public String toString()
        {
                return super.toString() + "\nlizardLength: " + lizardLength + "\nlizardLocation: "
+ lizardLocation;
        }

        public static void main(String a[])
```

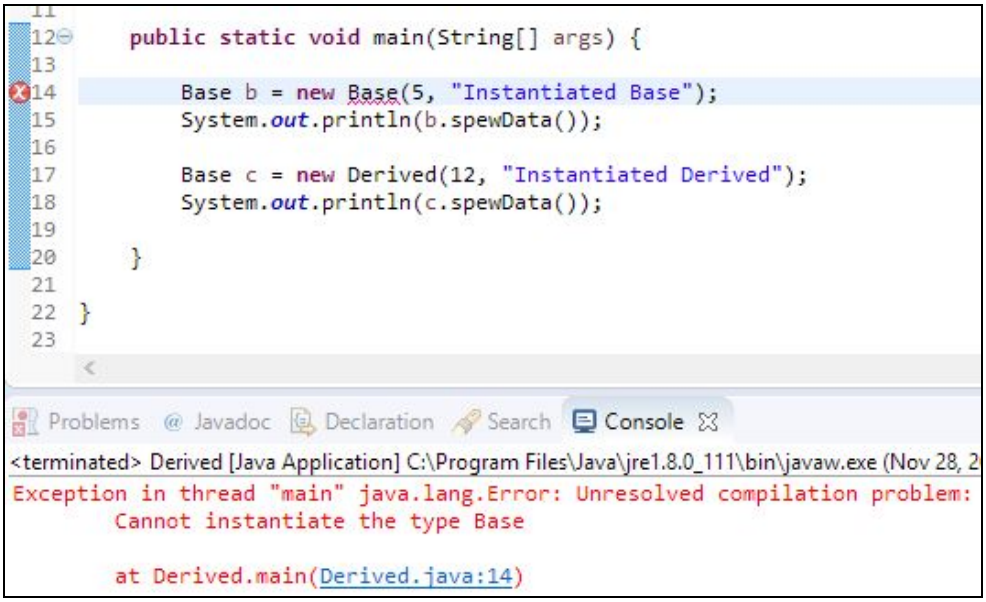
```
        {
                Lizard r = new Lizard(.9,4.0,10,"tropical");
                System.out.println(r);
        }
} // end class Lizard
```

3. Abstract Classes

Base Class:

```
2  public abstract class Base {
3
4      private int num;
5      private String term;
6
7      public Base (int n, String t) {
8          num = n;
9          term = t;
10     }
11
12     public abstract String spewData();
13
14     public int getNumber() {
15         return num;
16     }
17
18     public String getString() {
19         return term;
20     }
21 }
```

Trying to create Base:

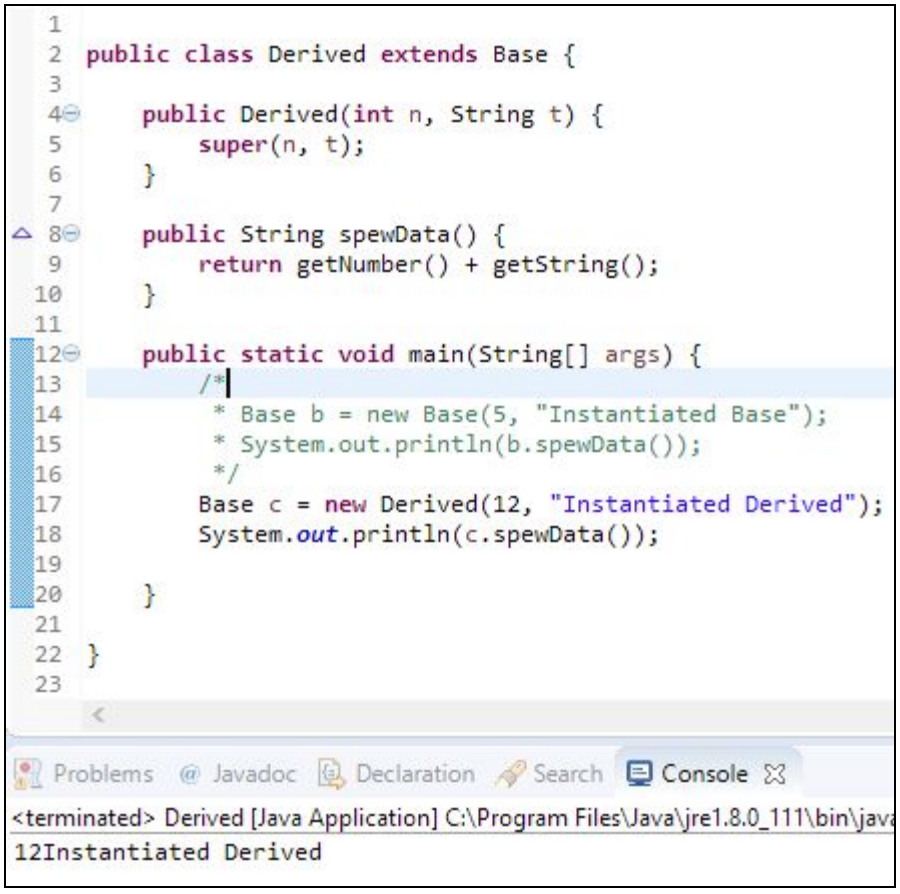


```
    public static void main(String[] args) {

        Base b = new Base(5, "Instantiated Base");
        System.out.println(b.spewData());

        Base c = new Derived(12, "Instantiated Derived");
        System.out.println(c.spewData());

    }

}
```

Problems @ Javadoc Declaration Search Console

```
<terminated> Derived [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Nov 28, 2
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot instantiate the type Base

        at Derived.main(Derived.java:14)
```

Trying to create Derived:



```
public class Derived extends Base {

    public Derived(int n, String t) {
        super(n, t);
    }

    public String spewData() {
        return getNumber() + getString();
    }

    public static void main(String[] args) {
        /*
         * Base b = new Base(5, "Instantiated Base");
         * System.out.println(b.spewData());
         */
        Base c = new Derived(12, "Instantiated Derived");
        System.out.println(c.spewData());

    }

}
```

Problems @ Javadoc Declaration Search Console

```
<terminated> Derived [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\java
12Instantiated Derived
```

4. Create a Variety of People

```
public class Person {
	private String name;

	public Person (String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}
}


public class Customer extends Person {
	private String address;

	public Customer (String name, String address) {
		super(name);
		this.address = address;
	}

	public String getAddress() {
		return address;
	}

	public String toString() {
		return "Name: " + getName() + "\nAddress: " + getAddress() + "\n\n";
	}
}


public class Employee extends Person {
	private int id;

	public Employee (String name, int id) {
		super(name);
		this.id = id;
	}

	public int getID () {
		return id;
	}
```

```
}


public class FullTime extends Employee {
	public double salary;

	public FullTime (String name, int id, double salary) {
		super(name, id);
		this.salary = salary;
	}

	public double getSalary () {
		return salary;
	}

	public String toString () {
		return "Name: " + getName() + "\nID: " + getID() + "\nSalary: " + salary + "\n\n";
	}
}


public class PartTime extends Employee {
	public double hourlyWage;

	public PartTime (String name, int id, double salary) {
		super(name, id);
		this.hourlyWage = salary;
	}

	public double getHourlyWage () {
		return hourlyWage;
	}

	public String toString () {
		return "Name: " + getName() + "\nID: " + getID() + "\nHourly Wage: " +
hourlyWage + "\n\n";
	}
}


public class HierarchyTest {
	public static void main(String[] args) {
		Employee e [] = new Employee[2];
		e[0] = new PartTime("Benny", 300, 15.45);
		e[1] = new FullTime("Jimbo", 90210, 32101.23);
```

```
                for (Employee emp : e) {
                        System.out.print(emp);
                }

                Person p [] = new Person[3];
                p[0] = new Customer("Thomas", "The Cloud");
                p[1] = new FullTime("Howie", 101, 50000.01);
                p[2] = new PartTime("Sodexo", 711, 8.56);

                for (Person per : p) {
                        System.out.print(per);
                }
        }
}
```

Output:

```
<terminated> HierarchyTest [J
Name: Benny
ID: 300
Hourly Wage: 15.45

Name: Jimbo
ID: 90210
Salary: 32101.23

Name: Thomas
Address: The Cloud

Name: Howie
ID: 101
Salary: 50000.01

Name: Sodexo
ID: 711
Hourly Wage: 8.56
```