UNIVERSITÉ Concordia UNIVERSITY

CENTRE FOR CONTINUING EDUCATION

INTRODUCTION TO PROGRAMMING WITH JAVA - CEJV416

Lecture #10

Composition and Inheritance

Static class variables

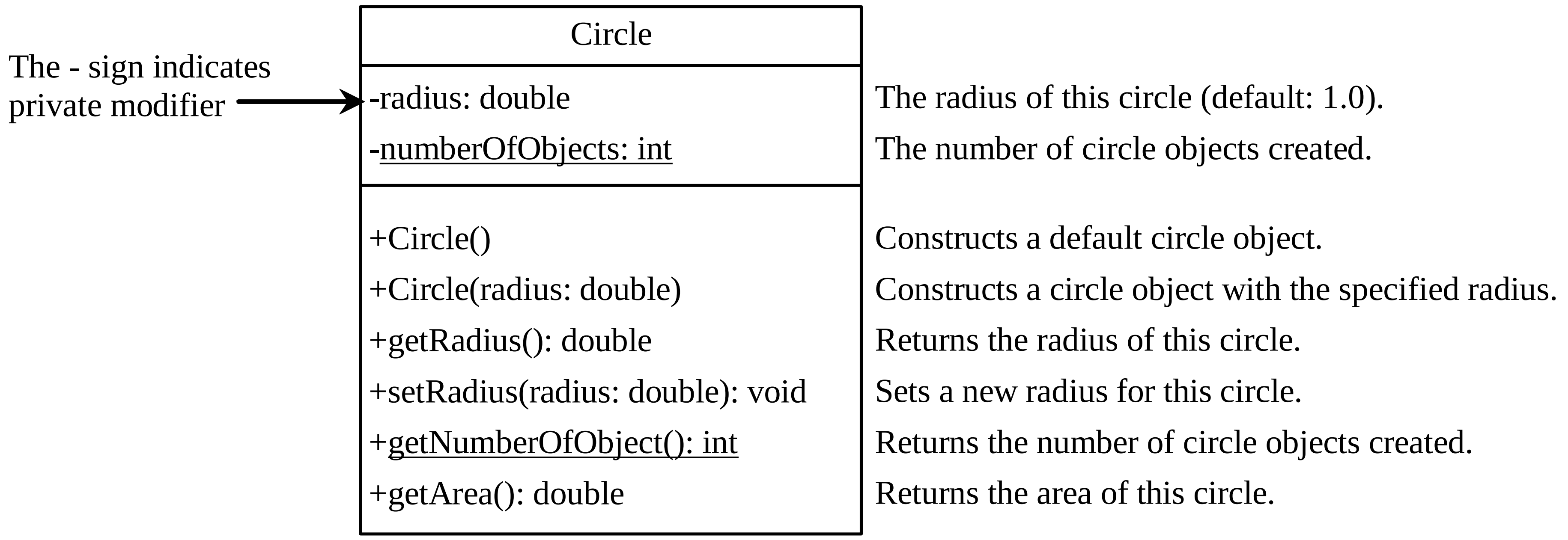
- These variables are shared by all objects of the same class
- If a value must be shared or if a change to a variable must be seen by all objects then make the field static
- The more common use is in creating constants

```
private static int numberOfObjects = 0;
private static double majorityPercent = .51;
public static final int DAYS_IN_JANUARY = 31;
public static final float EARTH_MASS_IN_KG = 5.972e24F;
```

Static methods and fields

- Use the static keyword to code *static fields* and *static methods.*
- Static fields and static methods belong to the class and not to an object created from the class
- Sometimes called *class fields* and *class methods*
- Static methods can only use static fields and fields that are defined in the method
- Cannot use instance variables in a static method because they belong to an instance of the class, not to the class as a whole

Example of Static Fields



-radius: double	The radius of this circle (default: 1.0).
-numberOfObjects: int	The number of circle objects created.
+Circle()	Constructs a default circle object.
+Circle(radius: double)	Constructs a circle object with the specified radius.
+getRadius(): double	Returns the radius of this circle.
+setRadius(radius: double): void	Sets a new radius for this circle.
+getNumberOfObject(): int	Returns the number of circle objects created.
+getArea(): double	Returns the area of this circle.

Exercise 19

- Rectangle
- Stock

Inheritance

has-a

- When one object contains another object we say that the containing object has an instance of the other object
- A Theatre class has an instance of a TheatreMap class
- This type of relationship is all about ownership
- It represents an excellent way to reuse existing classes

The Automobile

- Object oriented programming provides for another type of relationship
- If we were to describe a basic automobile we might build a class such as:

```
public class Automobile {
    private int wheels;
    private int doors;
    private Engine FourCylinder;
    private Radio am;
    private Seats bench;
    private Transmission manual;

    public Automobile() {..}
    public int getWheels() {..}
    public void setWheels(int w) {..}
    public int getDoors() {..}
    public void setDoors(int d){..}
    ...
}
```

A new car model

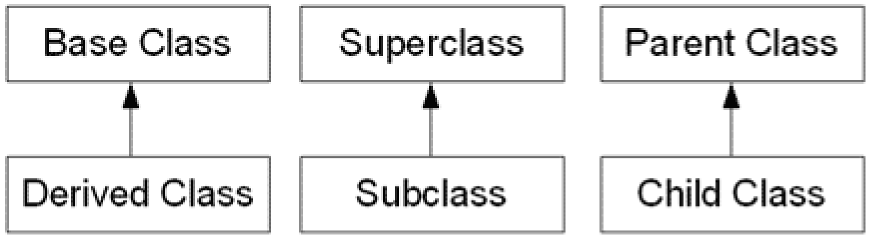
- Now I would like to introduce a new model of automobile
- This new car will have a convertible roof and just two doors
- We will name this model the ModelK!
- Given that we already have a basic car model, how should we describe the relationship between Automobile and ModelK?

is-a

- Should the ModelK contain an Automobile?
- Unlikely since the ModelK is just a specialization of the Automobile
- The ModelK is an Automobile
- It shares most of the attributes and behaviours or an Automobile
- What the ModelK does is provide a new meaning to existing behaviours, provide additional behaviours, and add new attributes
- But the ModelK is still an automobile

Inheritance

- Inheritance is a relationship such that a new class is created based upon an existing class
- All of the public and protected attributes of the original class are inherited by the new class
- The new class can then redefine what it has inherited and/or add new members
- This is also called a generalization-specialization relationship
- The general class is called the base or parent or super class
- The specialized class is called the derived or child or sub class



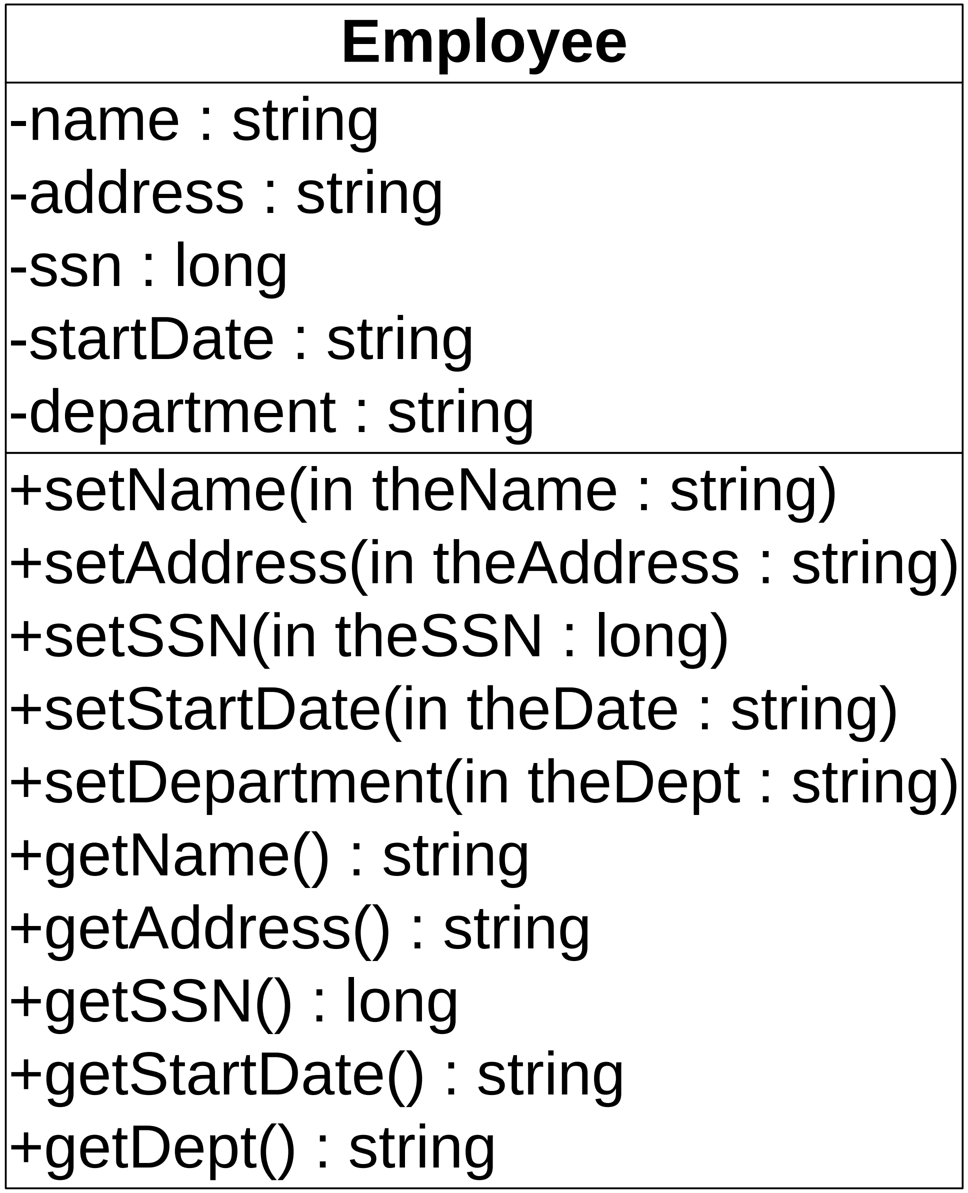
Human Resource Problem

- You have been assigned the task of developing classes to represent the employees in a company
- The company is a manufacturer of auto parts
- Employees fall into the following categories:
  - workers involved in the manufacturing process and paid an hourly rate
  - forepersons who direct the activities of groups and are paid an hourly rate
  - clerical staff working in the company's offices and paid a salary
  - salespersons who work for a base salary and a commission
  - managers who work for a salary and an annual bonus

Everyone is an Employee

- In analyzing the information you come to the realization that everyone working at this company is an employee
- The specific tasks and the method of remuneration may be different but they actually share more in common then they differ from one another
- This represents an excellent opportunity to apply the concepts of generalization/specialization

- We can identify attributes that are common to every employee in the company
- From this we build the general employee class

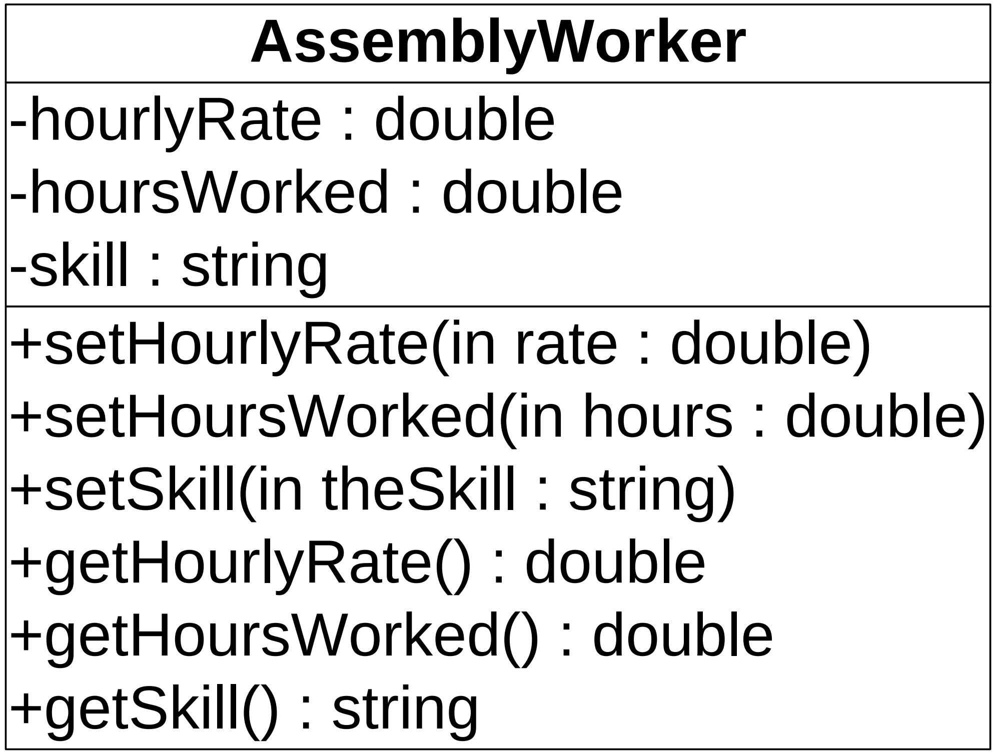


Employee

-name : string
-address : string
-ssn : long
-startDate : string
-department : string

+setName(in theName : string)
+setAddress(in theAddress : string)
+setSSN(in theSSN : long)
+setStartDate(in theDate : string)
+setDepartment(in theDept : string)
+getName() : string
+getAddress() : string
+getSSN() : long
+getStartDate() : string
+getDept() : string

- The employees who work in manufacturing on the assembly lines are called AssemblyWorker
- The attributes unique to this class of employee are their hourly rate, hours worked a week, and their skill

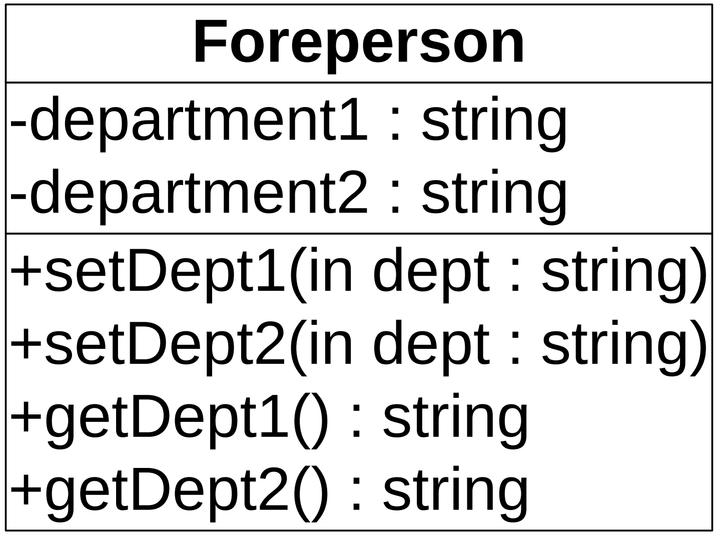


AssemblyWorker

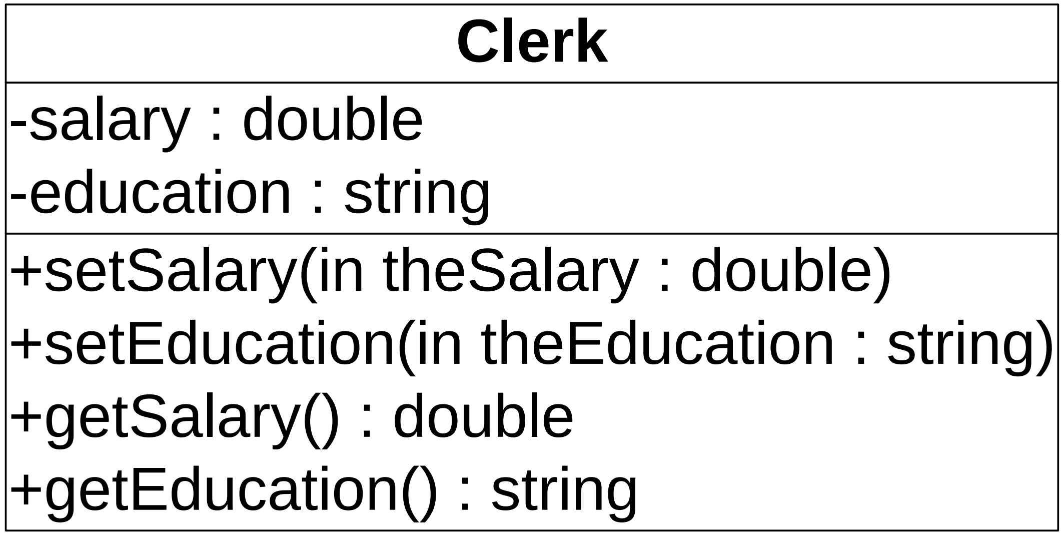
-hourlyRate : double
-hoursWorked : double
-skill : string

+setHourlyRate(in rate : double)
+setHoursWorked(in hours : double)
+setSkill(in theSkill : string)
+getHourlyRate() : double
+getHoursWorked() : double
+getSkill() : string

- AssemblyWorkers must be supervised and that is the job of a foreperson
- A Foreperson is a specialized AssemblyWorker who can supervise either one or two departments of workers



- A clerk is an employee who is paid an annual salary
- The education of this employee could have a bearing on their advancement in the company

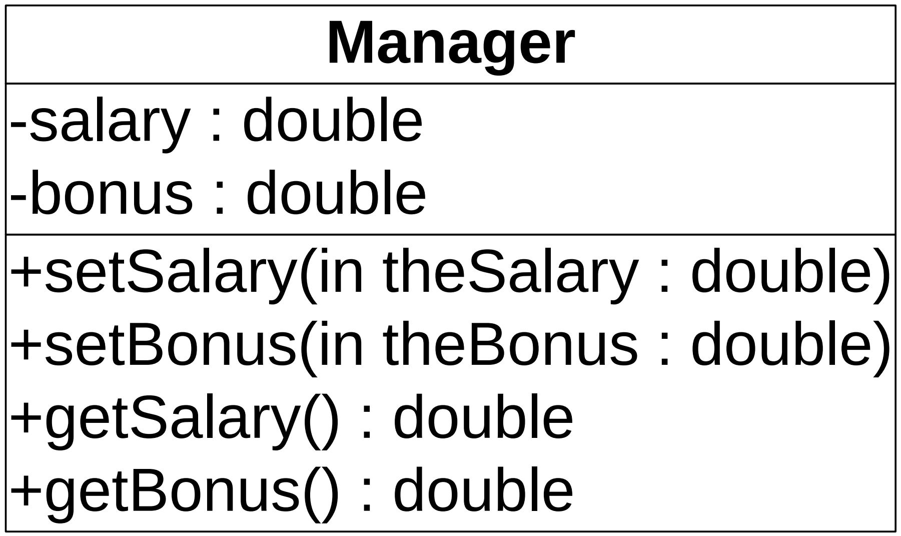


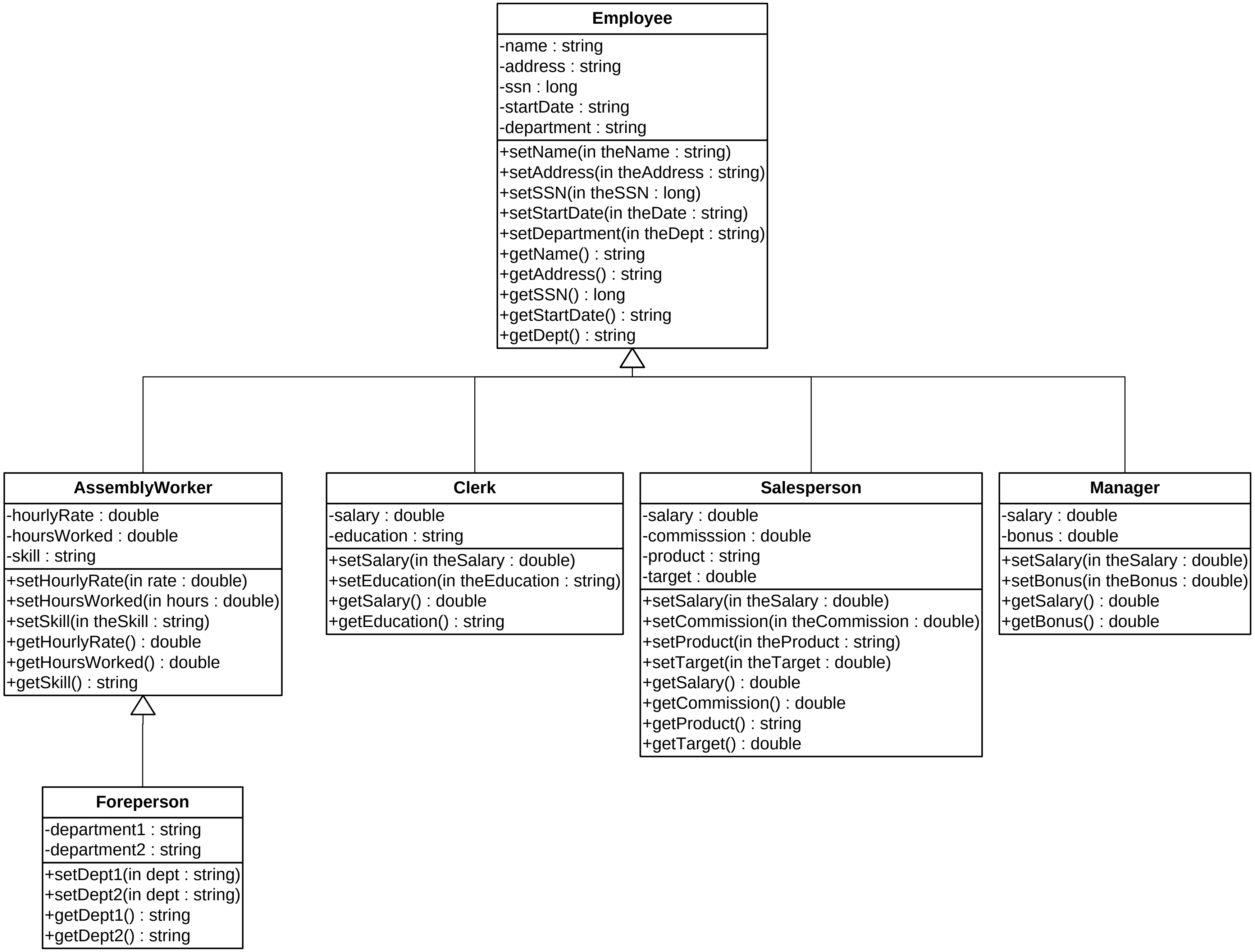
- A salesperson is paid a base salary
- A commission is paid when their sales exceeds the target assigned to them
- A salesperson is responsible for selling a single product

Salesperson
-salary : double -commisssion : double -product : string -target : double

+setSalary(in theSalary : double)
+setCommission(in theCommission : double)
+setProduct(in theProduct : string)
+setTarget(in theTarget : double)
+getSalary() : double
+getCommission() : double
+getProduct() : string
+getTarget() : double

- A manager is paid a salary plus an annual bonus




Employee
-name : string
-address : string
-ssn : long
-startDate : string
-department : string
+setName(in theName : string)
+setAddress(in theAddress : string)
+setSSN(in theSSN : long)
+setStartDate(in theDate : string)
+setDepartment(in theDept : string)
+getName() : string
+getAddress() : string
+getSSN() : long
+getStartDate() : string
+getDept() : string
AssemblyWorker
-hourlyRate : double
-hoursWorked : double
-skill : string
+setHourlyRate(in rate : double)
+setHoursWorked(in hours : double)
+setSkill(in theSkill : string)
+getHourlyRate() : double
+getHoursWorked() : double
+getSkill() : string
Clerk
-salary : double
-education : string
+setSalary(in theSalary : double)
+setEducation(in theEducation : string)
+getSalary() : double
+getEducation() : string
Salesperson
-salary : double
-commisssion : double
-product : string
-target : double
+setSalary(in theSalary : double)
+setCommission(in theCommission : double)
+setProduct(in theProduct : string)
+setTarget(in theTarget : double)
+getSalary() : double
+getCommission() : double
+getProduct() : string
+getTarget() : double
Manager
-salary : double
-bonus : double
+setSalary(in theSalary : double)
+setBonus(in theBonus : double)
+getSalary() : double
+getBonus() : double
Foreperson
-department1 : string
-department2 : string
+setDept1(in dept : string)
+setDept2(in dept : string)
+getDept1() : string
+getDept2() : string

```
public class Employee {
   private String name;
   private String address;
   private long ssn;
   private String startDate;
   private String department;

   public void setName(String name){this.name = name;}
   public void setAddress(String address)
       {this.address = address;}
   public void setSSN(long ssn){this.ssn = ssn;}
   public void setStartDate(String startDate )
       {this.startDate = startDate ;}
   public void setDepartment(String department)
       {this.department = department ;}
   public String getName(){return name;}
   public String getAddress(){return address;}
   public long getSSN(){return ssn;}
   public String getStartDate(){return startDate;}
   public String getDepartment(){return department;}
}
```

```
public class AssemblyWorker extends Employee{
   private double hourlyRate;
   private double hoursWorked;
   private String skill;

   public void setHourlyRate(double hourlyRate)
       {this.hourlyRate = hourlyRate;}
   public void setHoursWorked(double hoursWorked)
       {this.hoursWorked = hoursWorked;}
   public void setSkill(String skill )
      {this.skill = skill;}
   public double getHourlyRate()
      {return hourlyRate;}
   public double getHoursWorked()
      {return hoursWorked;}
   public String getSkill(){return skill;}
}
```

```
public class Foreperson extends AssemblyWorker{
   private String department1;
   private String department2;

   public void setDepartment1(String department1)
      {this.department1 = department1;}
   public void setDepartment2(String department2)
      {this.department2 = department2;}
   public String getDepartment1()
      {return department1;}
   public String getDepartment2()
      {return department2;}
}
```

```
public class Clerk extends Employee {
   private double salary;
   private String education;

   public void setSalary(double salary)
       {this.salary = salary;}
   public void setEducation(String education )
       {this.education = education ;}
   public double getSalary(){return salary;}
   public String getEducation(){return education;}
}
```

```
public class Salesperson extends Employee{
   private double salary;
   private double commission;
   private String product;
   private double target;

   public void setSalary(double salary)
        {this.salary = salary;}
   public void setCommission(double commission)
        {this.commission = commission;}
   public void setProduct(String product)
        {this.product = product;}
   public void setTarget(double target)
        {this.target = target;}
   public double getSalary(){return salary;}
   public double getCommission()
      {return commission;}
   public String getProduct(){return product;}
   public double getTarget(){return target;}
}
```

```
public class Manager extends Employee {
   private double salary;
   private double bonus;

   public void setSalary(double salary )
       {this.salary = salary ;}
   public void setBonus(double bonus )
      {this.bonus = bonus ;}
   public double getSalary(){return salary;}
   public double getBonus(){return bonus;}
}
```

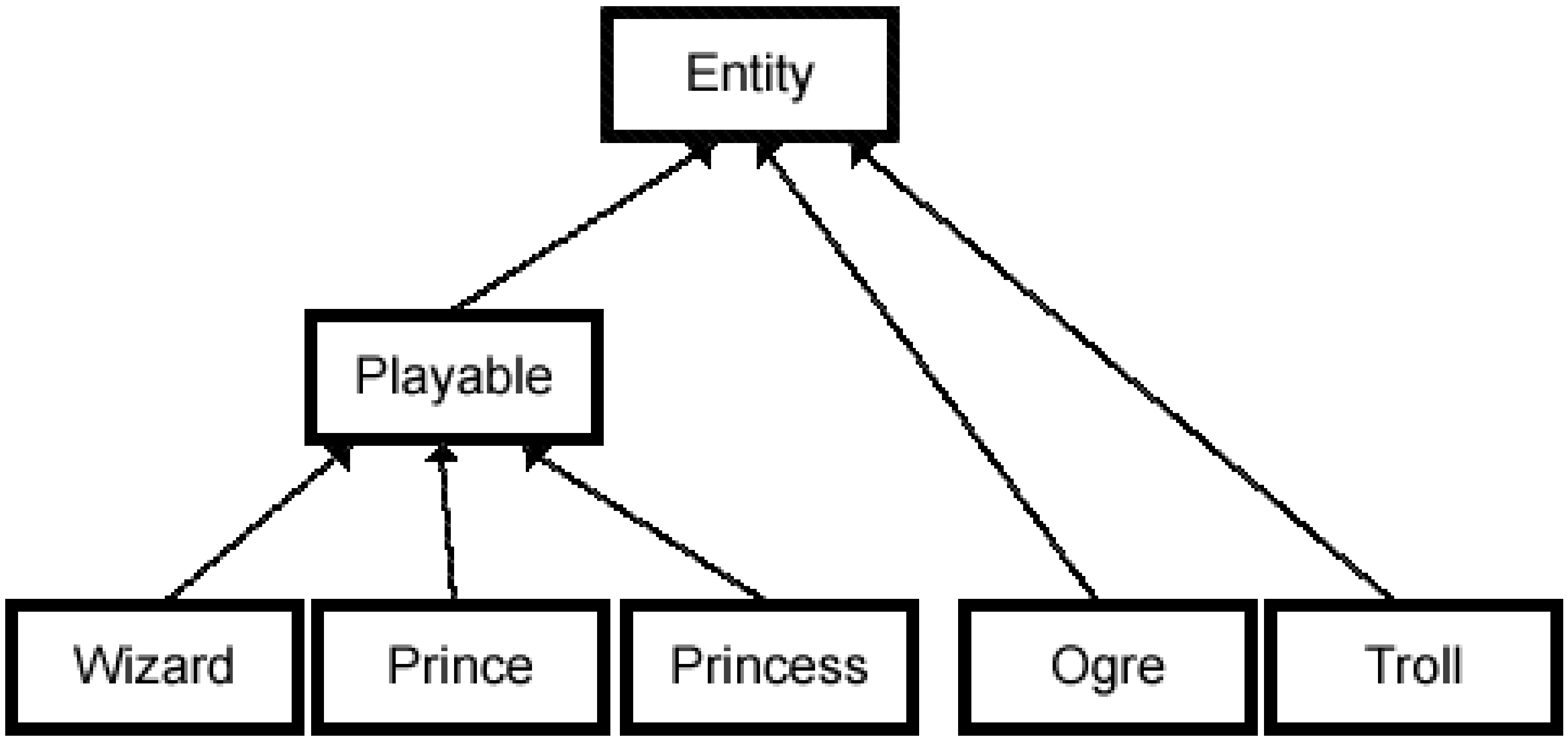
The Nature of the Inheritance Relationship

- Inheritance is about reusing the interface of a class
- The interface or public regions of a base class become part of the interface of the derived class
- The private region of the base class remains private
- In our example, to set the name of a Foreperson you would code:

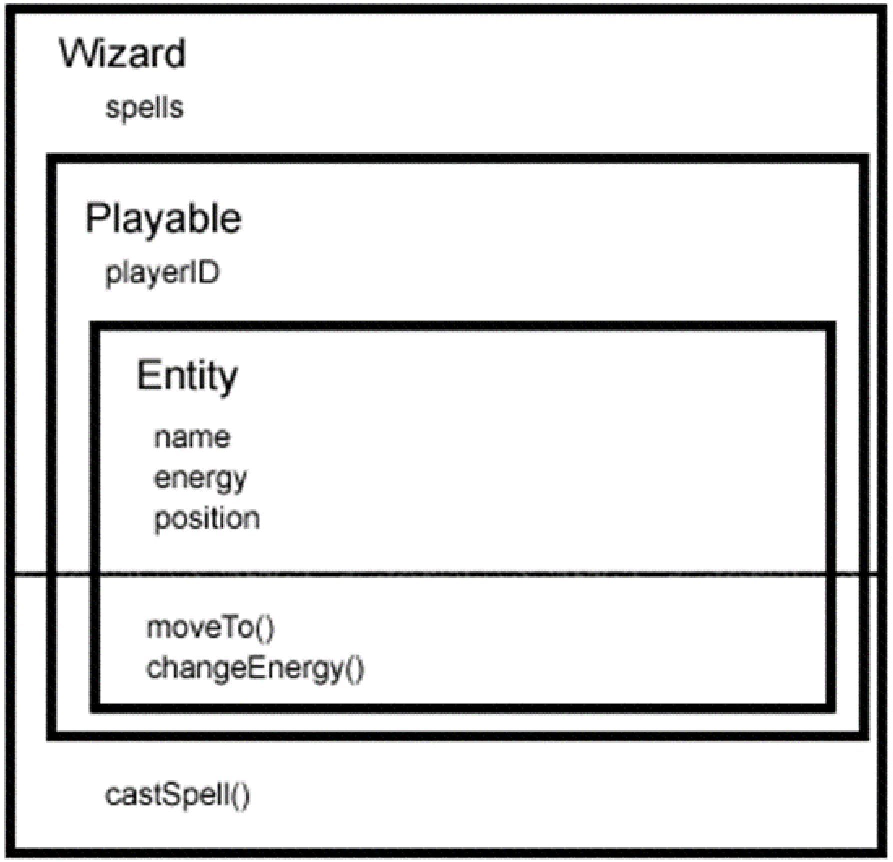
```
Foreperson fp = new Foreperson();
fp.setName("Bill Smith");
```

- The setName function is not part of Foreperson and not part of AssemblyWorker
- But it is a part of Employee from whom Foreperson is derived from

Game Example



Derived Class



Inheritance Concepts

- *Inheritance* lets you create a new class based on an existing class.
- The new class *inherits* the fields, constructors, and methods of the existing class.
- A class that inherits from an existing class is called a *derived class*, *child class,* or *subclass*.
- A class that another class inherits is called a *base class*, *parent class*, or *superclass*.
- A subclass can *extend* the superclass by adding new fields, constructors, and methods to the superclass.
- A subclass can *override* a method from the superclass with its own version of the method.

Derived Class

- You can directly access fields that have public or protected access in the superclass.
- You can extend the superclass by adding new fields, constructors, and methods.
- You can override methods in the superclass by coding methods that have the same signatures.
- You use the ***super*** keyword to call a constructor or method of the superclass. If necessary, you can call constructors or methods that pass arguments to the superclass.

The syntax for creating subclasses

To declare a subclass

```
public class SubclassName extends SuperClassName{}
```

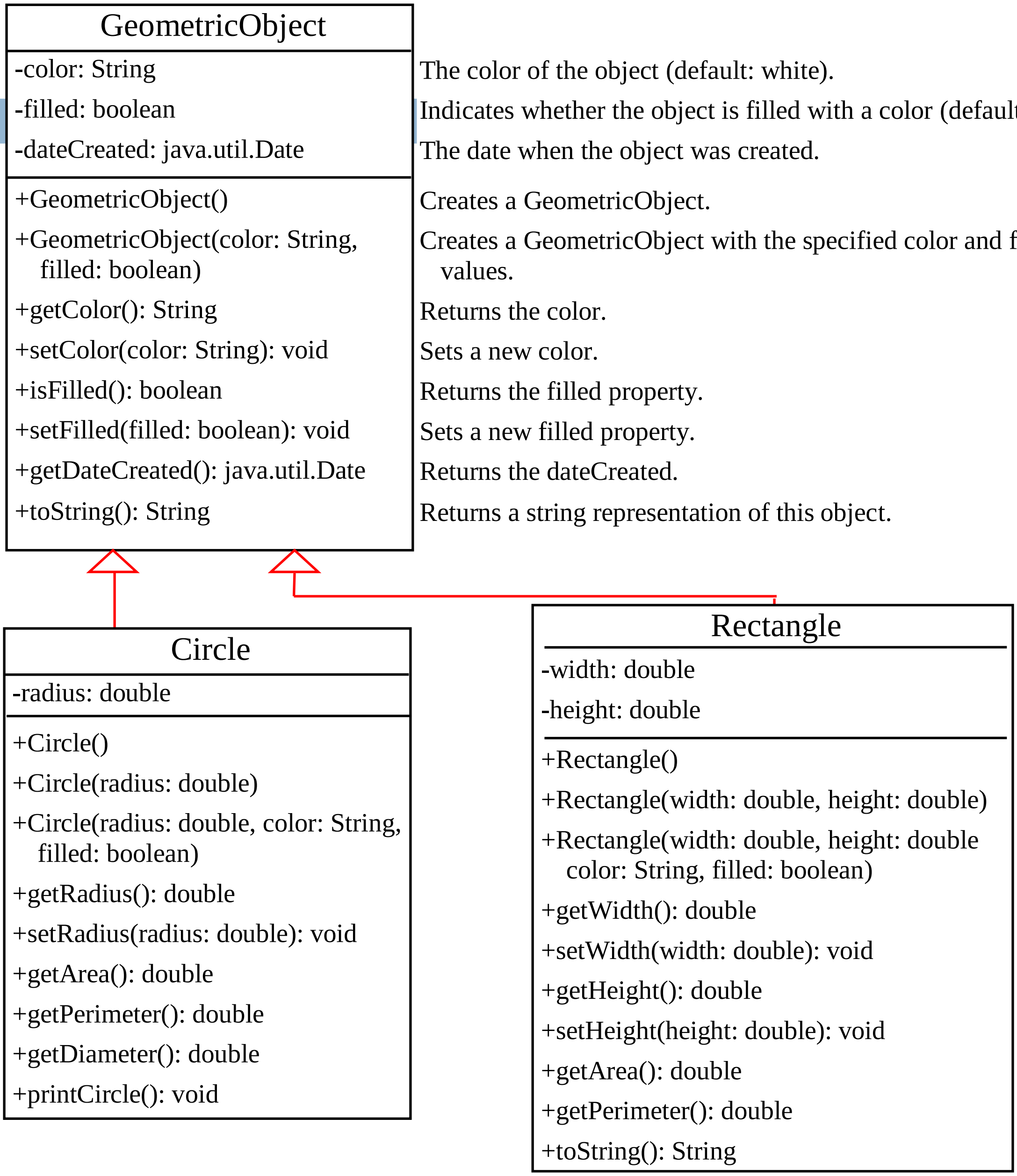
To call a superclass constructor

```
super(argumentList)
```

To call a superclass method

```
super.methodName(argumentList)
```

Exercise 20


GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
+GeometricObject()
+GeometricObject(color: String, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: String, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
+printCircle(): void
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double color: String, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double
+toString(): String

The color of the object (default: white).
Indicates whether the object is filled with a color (default: false).
The date when the object was created.
Creates a GeometricObject.
Creates a GeometricObject with the specified color and filled values.
Returns the color.
Sets a new color.
Returns the filled property.
Sets a new filled property.
Returns the dateCreated.
Returns a string representation of this object.