COMPSCI 121: INHERITANCE Part 2.

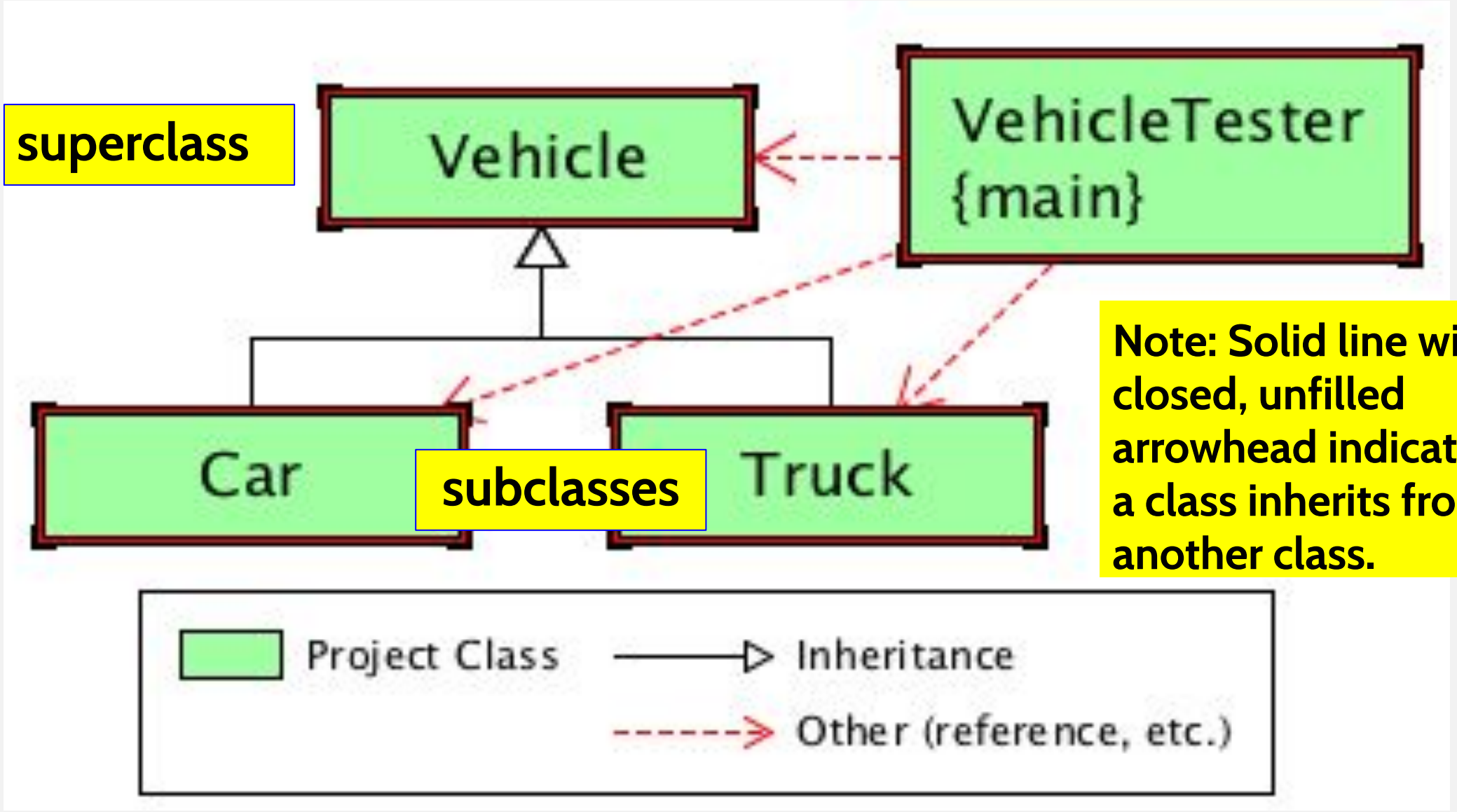
SPRING 2020

RECALL: INHERITANCE SUMMARY

We'll look at further aspects of inheritance, but first, a recap:

1. **Subclass** *inherits* from a **superclass**.
2. **Subclass** *adds its own special attributes*.
3. Subclass **constructor** calls its superclass constructor.
4. Subclass can **override** superclass methods to specialize them.
5. Subclass methods can call **super.method**() to do some of the work of the subclass version of the method.

FROM PREVIOUS LECTURE: INHERITANCE EXAMPLE 1



INHERITANCE TERMS

These are terms used to talk about inheritance:

- superclass, also called a base class.
- subclass, also called a derived class.
- About the relationship between superclass and subclss we can say:
  - “A subclass inherits from a superclass”.
  - “A subclass is derived from a superclass”.
  - “A subclass extends a superclass”.

USEFULNESS OF INHERITANCE

- **Inheritance** solves the duplication problem by "**refactoring**" all of the common attributes and methods into one superclass.
  - The subclasses also get to **specialize** any superclass methods (notice the `toString` method).
  - Easier to add new subclasses and troubleshoot.

Why is this useful?

- If we just had `Student`, `Faculty`, and `Employee` classes, they would all **duplicate some attributes**- `fName` and `lName`, and any methods related to these attributes.
- Code duplication means that any **changes have to be made in *more than one place***.
- Adding **new attributes is also more difficult**. This can ***lead to errors***.

INHERITANCE GUIDELINES

In **Subclasses** you can:

1. declare **new member variables** in the subclass that are **not** in the superclass.
2. write a **method** in the subclass that has the same signature as the one in the superclass, thus **overriding** it with a different implementation.
3. declare new methods in the subclass that are **not** in the superclass. These will not be available via a superclass reference, however.

ACCESS BY MEMBERS OF DERIVED CLASSES

Members of subclass have access to public members of the super class NOT to private members.

SPECIFIER	DESCRIPTION
private	Accessible by self.
protected	Accessible by self, derived classes, and other classes in the same package.
public	Accessible by self, derived classes, and everyone else.
no specifi	Accessible by self and other classes in the same package.

Best Practice: NEVER leave this unspecified!!

ACCESS BY MEMBERS OF DERIVED CLASSES

```
public class Person{
   private String fName;
   private String lName;
   private String address;
   private String email;
   private String phone;
```

Person member variables are declared private. Therefore, they are not directly accessible in the Employee subclass. For example:

```
public class Employee extends Person{

   public String toString(){
          return fName +"' "+lName+", "+hourlyRate;}
```

```
Employee.java:15: error: fName has private access in Person
          return fName +"' "+lName+", "+hourlyRate;}
                 ^
Employee.java:15: error: lName has private access in Person
          return fName +"' "+lName+", "+hourlyRate;}
                          ^
2 errors
```

ACCESS BY MEMBERS OF DERIVED CLASSES

Keyword `protected` provides access to derived classes and other classes in the same package (folder in which program files are located) but not by anyone else.

SPECIFIER	DESCRIPTION
private	Accessible by self.
protected	Accessible by self, derived classes, and other classes in the same package.
public	Accessible by self, derived classes, and everyone else.
~~no specifier~~	~~Accessible by self and other classes in the same package.~~

Best Practice: NEVER leave this unspecified!!

ACCESS BY MEMBERS OF DERIVED CLASSES

```
public class Person{
   protected String fName;
   protected String lName;
   protected String address;
   protected String email;
   protected String phone;
```

```
public class Employee extends Person{

   public String toString(){
          return fName +"' "+lName+", "+hourlyRate;}
```

Now, if Person member variables are declared protected, they ***are*** directly accessible in the Employee subclass. More on this coming up with the Business Project code.

UML: FROM JGRASP

Business

FIELDS
- address: protected java.lang.String address
- name: protected java.lang.String name

CONSTRUCTORS
- Business(): public Business()

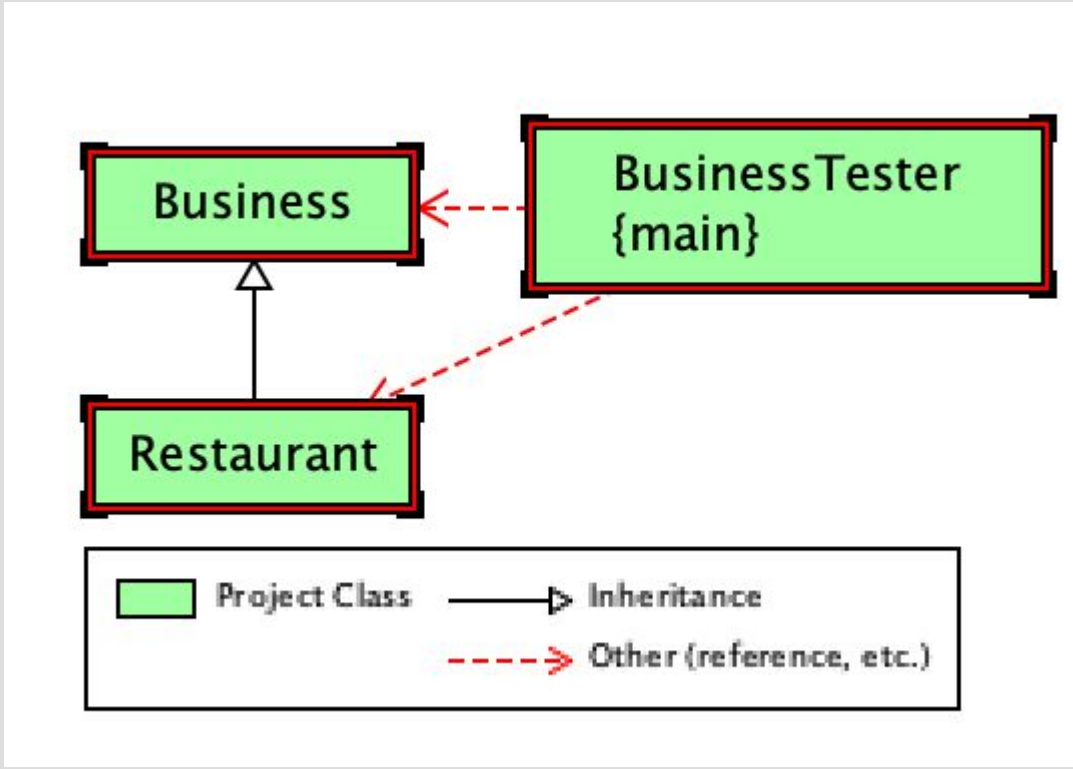
METHODS
- getDescription(): java.lang.String getDescription()
- setAddress(): void setAddress(java.lang.String)
- setName(): void setName(java.lang.String)
- toString(): public java.lang.String toString()

Restaurant

FIELDS
- rating: private int rating

CONSTRUCTORS
- Restaurant(): public Restaurant()

METHODS
- getRating(): public int getRating()
- setRating(): public void setRating(int)
- toString(): public java.lang.String toString()



Q: What methods does the `Restaurant` class inherit from `Business`? (A: all public methods)

UML: FROM JGRASP

Business

FIELDS
- address: protected java.lang.String address
- name: protected java.lang.String name

CONSTRUCTORS
- Business(): public Business()

METHODS
- getDescription(): java.lang.String getDescription()
- setAddress(): void setAddress(java.lang.String)
- setName(): void setName(java.lang.String)
- toString(): public java.lang.String toString()

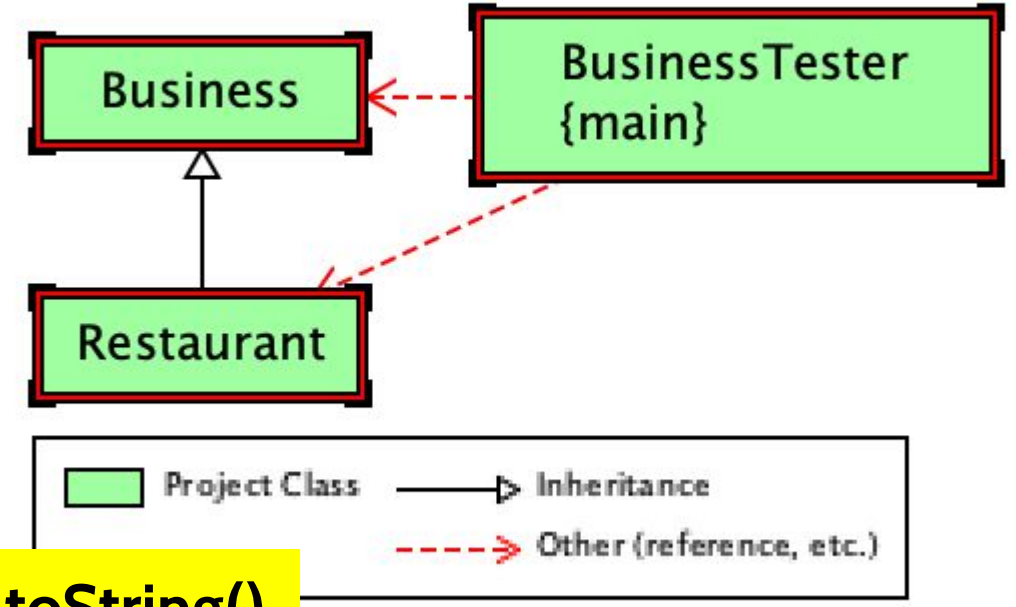
Restaurant

FIELDS
- rating: private int rating

CONSTRUCTORS
- Restaurant(): public Restaurant()

METHODS
- getRating(): public int getRating()
- setRating(): public void setRating(int)
- toString(): public java.lang.String toString()

Restaurant overrides toString()



Q: Business overrides the toString() method of which class?
A: java.lang.Object

OBJECT CLASS: toString() METHOD

java.lang

Class Object

java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. A

`String`	**`toString()`** Returns a string representation of the object.

Every class is a subclass of Object, so Business is overriding the toString() implementation of Object.

ACCESS BY MEMBERS OF DERIVED CLASSES

```
public class Business {

    @Override
    public String toString() {
        return name + " -- " + address;
    }
```

```
public class Restaurant extends Business {

    @Override
    public String toString() {
        return super.toString() + ", Rating: " + rating
    }
```

If we execute these statements in the BusinessTester class:

```
System.out.println(aaaBus.toString());
System.out.println(tacoRest.toString());
```

we get this output:

```
AAA Business -- 5 Race St
Tom's Tacos -- 600 Pleasure Ave, Rating: 5
```

OBJECT CLASS: toString() METHOD

```
1  public class Restaurant extends Business {
2      private int rating;
3
4      public void setRating(int userRating) {
5          rating = userRating;
6      }
7
8      public int getRating() {
9          return rating;
10     }
11
12     @Override
13     public String toString() {
14         return super.toString() + ", Rating: " + rating;
15     }
16 }
```

```
AAA Business -- 5 Race St
Tom's Tacos -- 600 Pleasure Ave, Rating: 5
```

`Restaurant` class `toString()` uses super keyword to call the base class `toString()` to get a string with the business `name` and `address`. Then `toString()` concatenates the `rating` and returns a string containing the `name`, `address`, and `rating`.

ACCESS BY MEMBERS OF DERIVED CLASSES

```
public class Business {

/*    @Override
    public String toString() {
        return name + " -- " + address;
    }  */
```

```
public class Restaurant extends Business {

/*    @Override
    public String toString() {
        return super.toString() + ", Rating: " + rating;
    }  */
```

If we remove the overrides, Java will execute Object's version of toString()...

```
System.out.println(aaaBus.toString());
System.out.println(tacoRest.toString());
```

... and we get this output:

```
Business@4d7e1886
Restaurant@3cd1a2f1
```

ACCESS BY MEMBERS OF DERIVED CLASSES

```
public class Business {

    @Override
    public String toString() {
        return name + " -- " + address;
    }
```

```
public class Restaurant extends Business {

/*    @Override
    public String toString() {
        return super.toString() + ", Rating: " + rat
    }  */
```

If we remove the Restaurant version of toString:

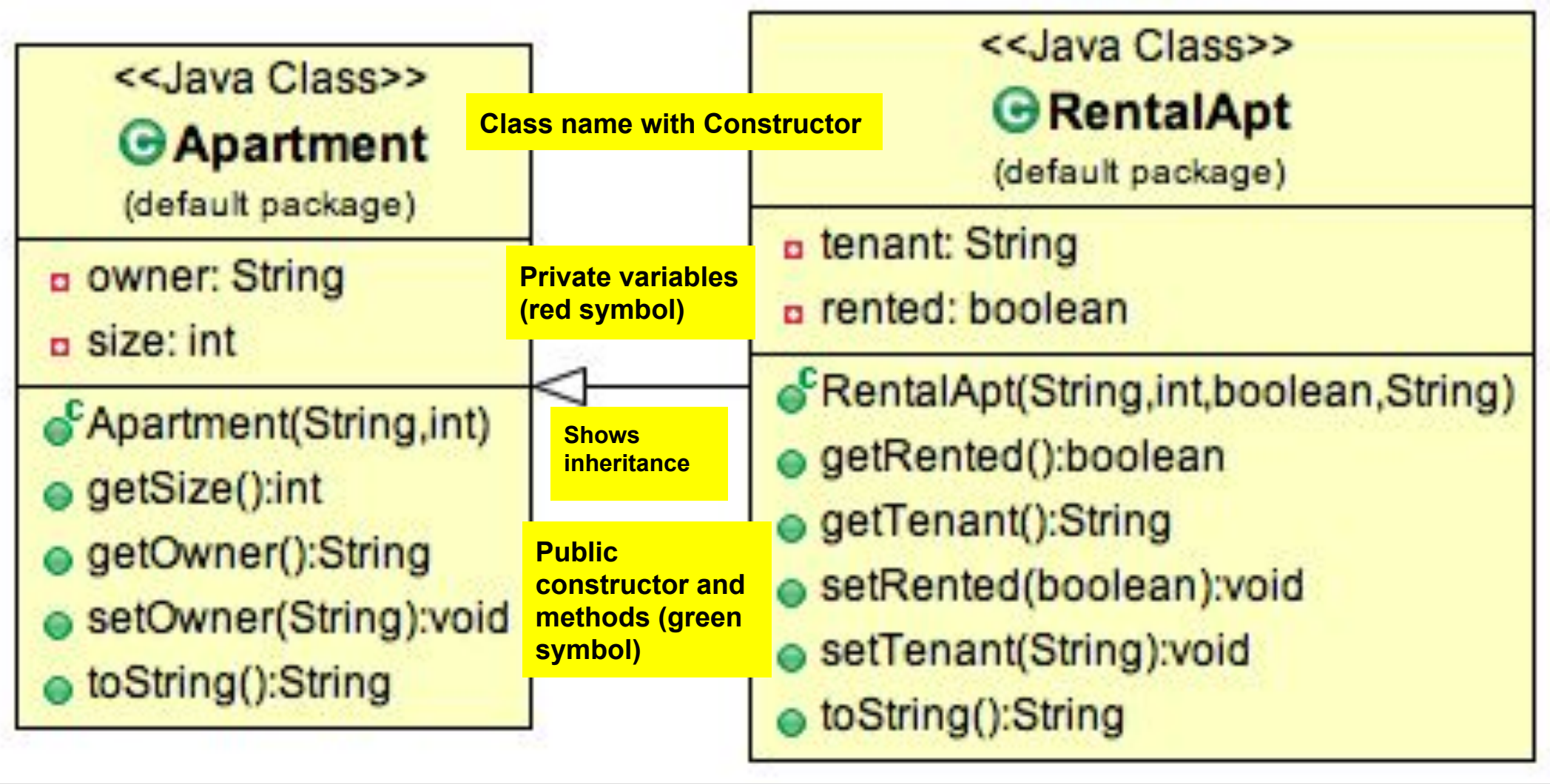
```
System.out.println(aaaBus.toString());
System.out.println(tacoRest.toString());
```

we get this output (notice the rating is missing):

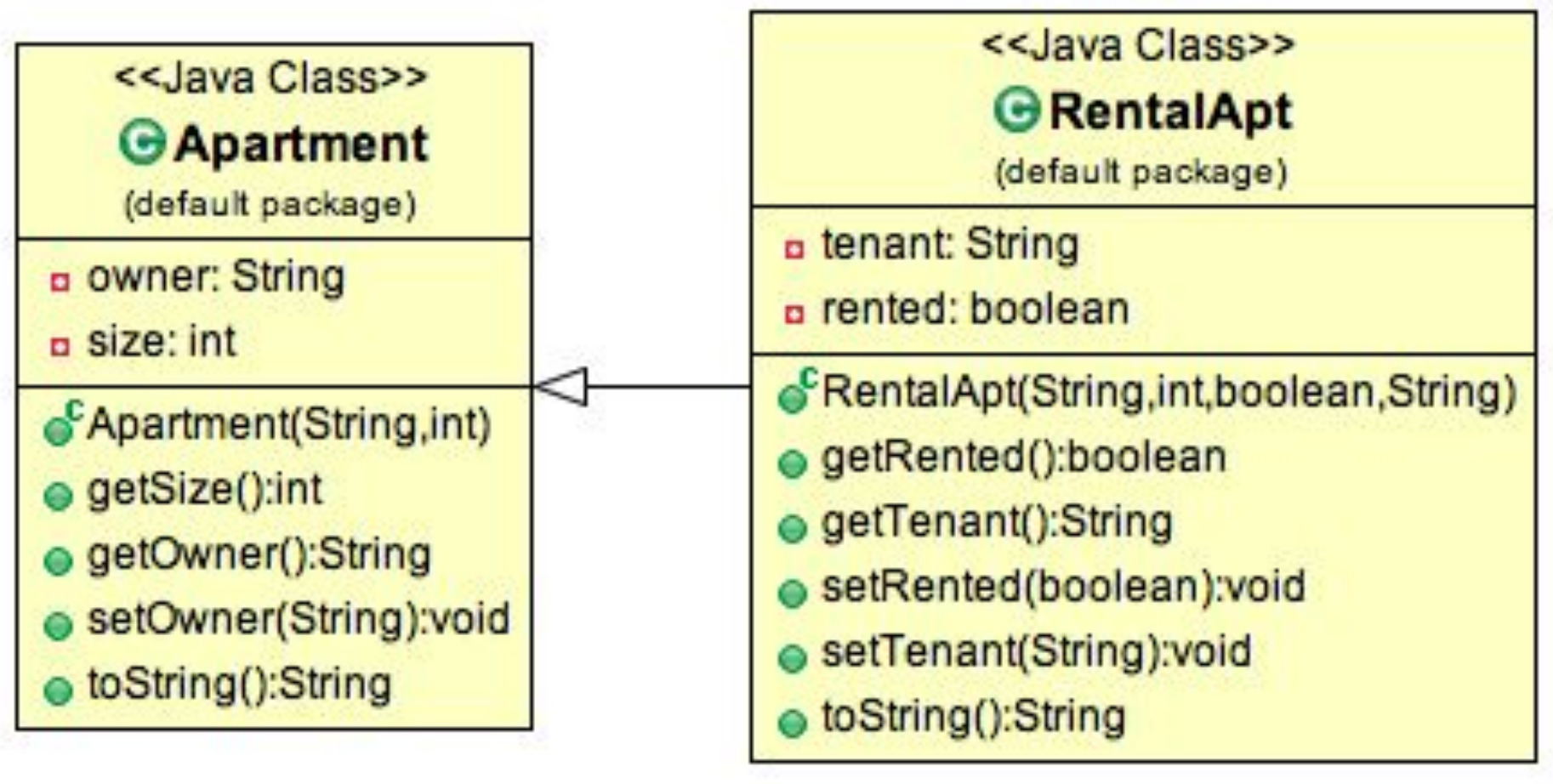
```
AAA Business -- 5 Race St
Tom's Tacos -- 600 Pleasure Ave
```

UML DIAGRAMS: APARTMENT EXAMPLE

Unified Modeling Language (UML) diagrams enable us to visually show classes and relationships between classes.

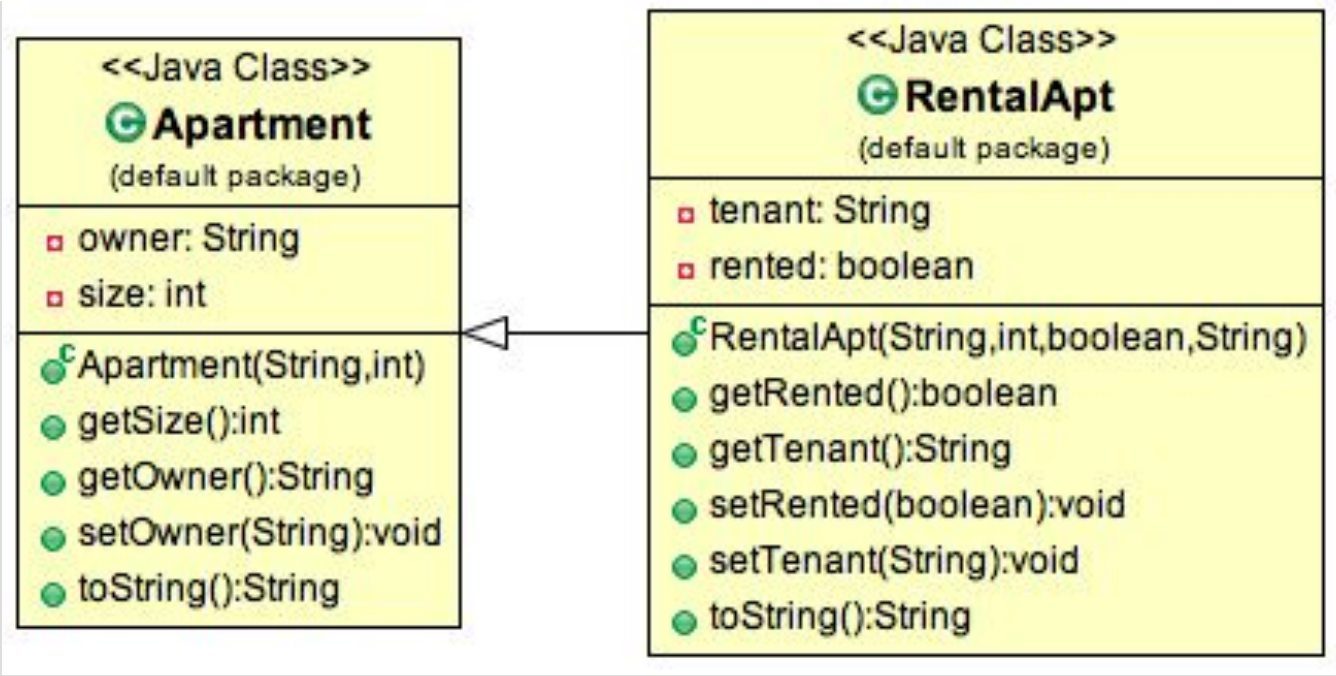


EXAMPLE 1 UML DIAGRAM



T-P-S
What are the inherited methods in RentalApt?

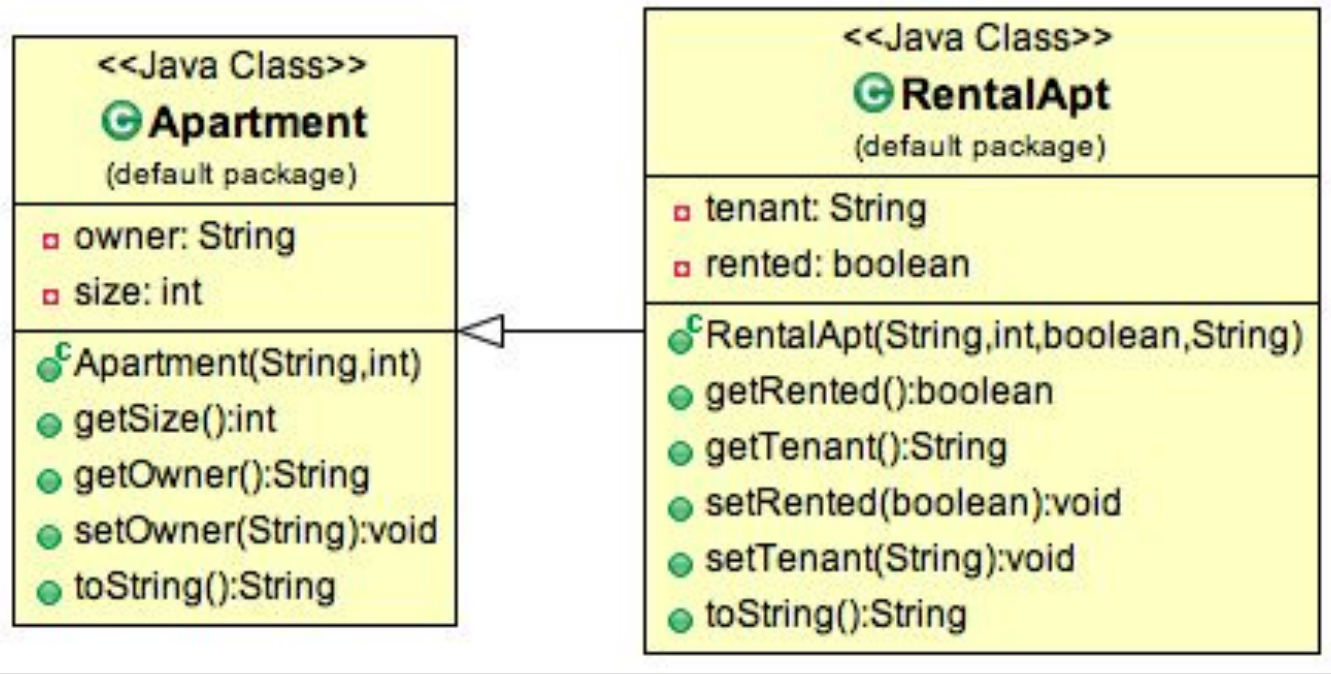
Clicker Question 1



RentalApt overrides

A. The `getSize()`, `getOwner()` methods
B. The `toString()` method
C. None of the super class methods
D. The `getRented()`, `getTenant()` methods
E. `RentalApt` is not a subclass of `Apartment`

Clicker Question 1 Answer



RentalApt overrides

A. The `getSize()`, `getOwner()` methods
B. The `toString()` method
C. None of the super class methods
D. The `getRented()`, `getTenant()` methods
E. `RentalApt` is not a subclass of `Apartment`

WEEK 10 TO DO

- Complete zyBook chapter 10.
- Start the next project early - good for practising concepts.
- Visit online office hours for help.
- Post in Moodle or Piazza for help.