

2023-02-24

9.2: Inheritance and Constructors

Java & Inheritance

Recall - Subclasses **inherit** all the variables and methods of their superclass

```
class Person {  
    public String name;  
    public String address;  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}
```

```
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person();  
p.name = "Gary";  
p.address = "San Francisco";  
p.printInfo();
```

```
Teacher t = new Teacher();  
t.name = "Chris";  
t.address = "San Mateo";  
t.printInfo();  
t.office = "215W";
```

Java & Inheritance

Note: Not a best practice to make these public (more on this in a couple of slides)

Recall - Subclasses **inherit** all the variables and methods of their superclass

```
class Person {  
    public String name;  
    public String address;  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person();  
p.name = "Gary";  
p.address = "San Francisco";  
p.printInfo();
```

```
Teacher t = new Teacher();  
t.name = "Chris";  
t.address = "San Mateo";  
t.printInfo();  
t.office = "215W";
```

Java & Inheritance

Subclasses can only access the **public** variables and **public** methods of their superclass

```
class Person {
    public String name;
    public String address;
    public void printInfo() {
        String info = buildInfoString();
        System.out.println(info);
    }
    private String buildInfoString() {
        return name + " " + address;
    }
}

class Teacher extends Person {
    public String office;
    public String getBuildInfoString() {
        return buildInfoString(); ** ERROR **
    }
}
```

```
Person p = new Person();
p.name = "Gary";
p.address = "San Francisco";
p.printInfo();

Teacher t = new Teacher();
t.name = "Chris";
t.address = "San Mateo";
t.printInfo();
t.buildInfoString(); ** ERROR **
t.office = "215W";
```

Java & Inheritance

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

Java & Inheritance

Note: This is the better practice - Keep your variables `private` unless you want other code to mess with 'em!

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

Java & Inheritance

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```


Java & Inheritance

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}
```

```
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

```
error: constructor Person in class Person cannot be applied to given  
types;  
class Teacher extends Person {  
^  
    required: String,String  
    found:    no arguments  
    reason: actual and formal argument lists differ in length
```

Java & Inheritance

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

Reminder: If you declare any Constructor, Java will no longer automatically create a no-param constructor for you.

In this example, since Person has a Constructor that requires two parameters, there is no way to create a Person with zero parameters.

And this error is telling you that Teacher is malformed because there is no way to properly create its Person superclass.

```
error: constructor Person in class Person cannot be applied to given  
types;  
class Teacher extends Person {  
^  
    required: String,String  
    found:    no arguments  
    reason: actual and formal argument lists differ in length
```

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person() {  
        // empty  
    }  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person() {  
        // empty  
    }  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person() {
        // empty
    }
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person() {
        // empty
    }
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

But did this really fix the issue?

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person() {
        // empty
    }
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

But did this really fix the issue?

Q: What is the output of t.printInfo()?

A:

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person() {  
        // empty  
    }  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

```
Teacher t = new Teacher();  
t.printInfo();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

But did this really fix the issue?

Q: What is the output of t.printInfo()?

A: null null

Java & Inheritance

Solution 1: Add a no-param constructor to Person

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person() {
        // empty
    }
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Adding a no-param constructor to Person "fixes" the error - Java now has a way to create a Teacher and its Person superclass.

But did this really fix the issue?

Q: What is the output of t.printInfo()?

A: null null

So let's try something else...

Java & Inheritance

Solution 2: Call the Person constructor from a Teacher constructor using `super()`

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher() {
        super("<a name>", "<an address>");
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();

Teacher t = new Teacher();
t.printInfo();
```

Java & Inheritance

Solution 2: Call the Person constructor from a Teacher constructor using `super()`

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher() {
        super("<a name>", "<an address>");
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Subclasses can invoke a constructor in their superclass with `super()`

- 1) `super()` may only be used on the first line of a subclass constructor
- 2) the params you pass to `super()` determine which superclass constructor is invoked

Java & Inheritance

Solution 2: Call the Person constructor from a Teacher constructor using `super()`

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher() {
        super("<a name>", "<an address>");
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Subclasses can invoke a constructor in their superclass with `super()`

- 1) `super()` may only be used on the first line of a subclass constructor
- 2) the params you pass to `super()` determine which superclass constructor is invoked

Q: Now what is the output of `t.printInfo()`?

A:

Java & Inheritance

Solution 2: Call the Person constructor from a Teacher constructor using `super()`

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher() {
        super("<a name>", "<an address>");
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Subclasses can invoke a constructor in their superclass with `super()`

- 1) `super()` may only be used on the first line of a subclass constructor
- 2) the params you pass to `super()` determine which superclass constructor is invoked

Q: Now what is the output of `t.printInfo()`?

A: <a name> <an address>

Java & Inheritance

Solution 2: Call the Person constructor from a Teacher constructor using `super()`

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher() {
        super("<a name>", "<an address>");
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher();
t.printInfo();
```

Subclasses can invoke a constructor in their superclass with `super()`

- 1) `super()` may only be used on the first line of a subclass constructor
- 2) the params you pass to `super()` determine which superclass constructor is invoked

Q: Now what is the output of `t.printInfo()`?

A: <a name> <an address>

Better - But probably still not the best solution...

Java & Inheritance

**Solution 3: Invoke super()
from a 2-param Teacher
constructor**

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher(String name, String address) {
        super(name, address);
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();

Teacher t = new Teacher("Chris", "Thilgen");
t.printInfo();
```


Java & Inheritance

**Solution 3: Invoke super()
from a 2-param Teacher
constructor**

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher(String name, String address) {
        super(name, address);
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher("Chris", "San Mateo");
t.printInfo();
```

Q: Now what is the output of t.printInfo()?

A:

Java & Inheritance

Solution 3: Invoke super() from a 2-param Teacher constructor

But what happens if items in the superclass are not public?

```
class Person {
    private String name;
    private String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }
    public void printInfo() {
        System.out.println(name + " " + address);
    }
}

class Teacher extends Person {
    public String office;
    public Teacher(String name, String address) {
        super(name, address);
    }
}
```

```
Person p = new Person("Gary", "San Francisco");
p.printInfo();
```

```
Teacher t = new Teacher("Chris", "San Mateo");
t.printInfo();
```

Q: Now what is the output of t.printInfo()?

A: Chris San Mateo

Java & Inheritance

**Solution 3: Invoke super()
from a 2-param Teacher
constructor**

But what happens if items in the superclass are not public?

```
class Person {  
    private String name;  
    private String address;  
    public Person(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
    public void printInfo() {  
        System.out.println(name + " " + address);  
    }  
}  
  
class Teacher extends Person {  
    public String office;  
    public Teacher(String name, String address) {  
        super(name, address);  
    }  
}
```

```
Person p = new Person("Gary", "San Francisco");  
p.printInfo();
```

```
Teacher t = new Teacher("Chris", "San Mateo");  
t.printInfo();
```

Q: Now what is the output of t.printInfo()?

A: Chris San Mateo

Huzzah!

Java & Inheritance - Access Modifiers

- We have previously discussed the `private` and `public` keywords (Access Modifiers) and how they are used inside a class to block or allow access to internal methods and variables to code outside the class
- `protected` is another Access Modifier that can be used to allow access to internal methods and variables to **subclasses** of the class (so it is mostly like `private` except for subclasses)

```
class Person {  
    public String name;  
    protected String age;  
    private String taxId;  
}  
  
class Teacher extends Person {}  
  
class Pet {}
```

	Can Access	Cannot Access
Person	?	?
Teacher	?	?
Pet	?	?

Java & Inheritance - Access Modifiers

- We have previously discussed the `private` and `public` keywords (Access Modifiers) and how they are used inside a class to block or allow access to internal methods and variables to code outside the class
- `protected` is another Access Modifier that can be used to allow access to internal methods and variables to **subclasses** of the class (so it is mostly like `private` except for subclasses)

```
class Person {  
    public String name;  
    protected String age;  
    private String taxId;  
}  
  
class Teacher extends Person {}  
  
class Pet {}
```

	Can Access	Cannot Access
Person	Person.name Person.age Person.taxId	
Teacher	?	?
Pet	?	?

Java & Inheritance - Access Modifiers

- We have previously discussed the `private` and `public` keywords (Access Modifiers) and how they are used inside a class to block or allow access to internal methods and variables to code outside the class
- `protected` is another Access Modifier that can be used to allow access to internal methods and variables to **subclasses** of the class (so it is mostly like `private` except for subclasses)

```
class Person {  
    public String name;  
    protected String age;  
    private String taxId;  
}  
  
class Teacher extends Person {}  
  
class Pet {}
```

	Can Access	Cannot Access
Person	Person.name Person.age Person.taxId	
Teacher	Person.name Person.age	Person.taxId
Pet	?	?

Java & Inheritance - Access Modifiers

- We have previously discussed the `private` and `public` keywords (Access Modifiers) and how they are used inside a class to block or allow access to internal methods and variables to code outside the class
- `protected` is another Access Modifier that can be used to allow access to internal methods and variables to **subclasses** of the class (so it is mostly like `private` except for subclasses)

```
class Person {  
    public String name;  
    protected String age;  
    private String taxId;  
}  
  
class Teacher extends Person {}  
  
class Pet {}
```

	Can Access	Cannot Access
Person	Person.name Person.age Person.taxId	
Teacher	Person.name Person.age	Person.taxId
Pet	Person.name	Person.age Person.taxId

Java & Inheritance - Access Modifiers

- We have previously discussed the `private` and `public` keywords (Access Modifiers) and how they are used inside a class to block or allow access to internal methods and variables to code outside the class
- `protected` is another Access Modifier that can be used to allow access to internal methods and variables to **subclasses** of the class (so it is mostly like `private` except for subclasses)

```
class Person {  
    public String name;  
    protected String  
    private String ta  
}  
  
class Teacher exten  
  
class Pet {}
```

Learn more about Access
Modifiers in the [Java
Documentation](#)

Person

Can Access

Cannot Access

Person.name
Person.age
Person.taxId

Teacher

Person.name
Person.age

Person.taxId

Pet

Person.name

Person.age
Person.taxId

Java & Inheritance

Summary

- Subclasses do not have access to the private variables and private methods of their superclass (create accessor methods or carefully decorate items with `protected`)
- Subclasses can use the `super()` function to invoke superclass constructors
- `super()` can only be used on the first line of a subclass constructor (to prevent subclasses from interfering with the creation of the superclass)
- The params passed to `super()` determine which constructor in the superclass is invoked
- If you do not add a call to `super()` in a subclass constructor - Java will automatically add call to `super()` with no params - i.e. it will call the no-param constructor of the superclass (to ensure that the super-class is properly created; because the subclass depends on it)

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w1 = new Widget();  
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w1 = new Widget();  
> WidgetA
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w2 = new Widget("some string");  
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w2 = new Widget("some string");  
> WidgetB
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w3 = new Widget(400, 100);  
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}
```

```
Widget w3 = new Widget(400, 100);  
> WidgetC
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget {  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ?
```


Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget {  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ""
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> WidgetA
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
    public SuperWidget() {  
        System.out.println("SuperWidgetA");  
    }  
}
```

```
SuperWidget sp1 = new SuperWidget();
```

```
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
    public SuperWidget() {  
        System.out.println("SuperWidgetA");  
    }  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> WidgetA  
> SuperWidgetA
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
    public SuperWidget() {  
        System.out.println("SuperWidgetA");  
        super();  
    }  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ?
```

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
    public SuperWidget() {  
        System.out.println("SuperWidgetA");  
        super();  
    }  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ?
```

ERROR: `super()` can only be used on the first line of a constructor!

Java & Inheritance

```
class Widget {  
    public Widget() {  
        System.out.println("WidgetA");  
    }  
    public Widget(String s) {  
        System.out.println("WidgetB");  
    }  
    public Widget(int x, int y) {  
        System.out.println("WidgetC");  
    }  
}  
  
class SuperWidget extends Widget {  
    public SuperWidget() {  
        super("some string");  
        System.out.println("SuperWidgetA");  
    }  
}
```

```
SuperWidget sp1 = new SuperWidget();  
> ?
```


Java & Inheritance

```
class Widget {
    public Widget() {
        System.out.println("WidgetA");
    }
    public Widget(String s) {
        System.out.println("WidgetB");
    }
    public Widget(int x, int y) {
        System.out.println("WidgetC");
    }
}

class SuperWidget extends Widget {
    public SuperWidget() {
        super("some string");
        System.out.println("SuperWidgetA");
    }
}
```

```
SuperWidget sp1 = new SuperWidget();
> WidgetB
> SuperWidgetA
```

Practice on your own

- CSAwesome 9.2 - Inheritance and Constructors
- Replit - Date2

Practice on your own

- CSAwesome 9.2 - Inheritance and Constructors
- Replit - Date2



<code>DateUtil.isDateWeekend(Date d)</code>	<code>d2.isDateWeekend()</code>
<code>DateUtil.hasCaseOfTheMondays(Date d)</code>	<code>d2.hasCaseOfTheMondays()</code>
<code>DateUtil.isDateLeapYear(Date d)</code>	<code>d2.isDateLeapYear()</code>
<code>DateUtil.isDateH1(Date d)</code>	<code>d2.isDateH1()</code>
<code>DateUtil.getDateInfo(Date d)</code>	<code>d2.getDateInfo()</code>

Practice on your own

- CSAwesome 9.2 - Inheritance and Constructors
- Replit - Date2



<code>DateUtil.isDateWeekend(Date d)</code>	<code>d2.isDateWeekend()</code>
<code>DateUtil.hasCaseOfTheMondays(Date d)</code>	<code>d2.hasCaseOfTheMondays()</code>
<code>DateUtil.isDateLeapYear(Date d)</code>	<code>d2.isDateLeapYear()</code>
<code>DateUtil.isDateH1(Date d)</code>	<code>d2.isDateH1()</code>
<code>DateUtil.getDateInfo(Date d)</code>	<code>d2.getDateInfo()</code>