Lecture 25

- Covers
  - Constructors
  - More information hiding
  - Packages
- Reading: Savitch 5.5, 5.6, 5.7

Lecture overview

- Default and Multiple Constructors
- Privacy Leaks
- Packages

- Default and multiple constructors

Constructors (recall)

- Methods called when a new object is created
- Can perform any type of statement
- Meant to be used to initialise the state of the object being created
- Called automatically by the new operator
- Same name as the class

Points of the section

- A class can have zero or more constructors
- When a class does not have a constructor defined for it, a default constructor will be used to create new objects
- When a class has more than one constructor their signatures have to be different, i.e. they must have different parameter type sequences
- The this keyword can be used in a constructor to refer to another constructor defined in the same class

Example

```
public class DigitalClock
{
    private int hours;
    private int minutes;

    public DigitalClock( )
    {
        hours = 0;
        minutes = 0;
    }
    …
```

Constructors

- Can take varying numbers and types of parameters
- Constructors that take no arguments are called default constructors

Example

```
public DigitalClock(int h, int m)
{
    setHours(h);
    setMinutes(m);
}
```

Constructors

- When creating an object, we must call the appropriate constructor

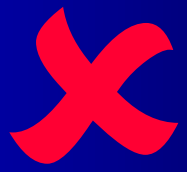
```
DigitalClock dc1 = new DigitalClock( );
DigitalClock dc2 = new DigitalClock(12, 23);
```

Constructors

- Constructors can only be invoked during instantiation of the object

- They cannot be invoked on existing objects

```
dc1.DigitalClock(23, 59);
```



Constructors

- Constructors return a reference to the new object (the new object's memory address)
- This address (reference) can then be stored as the value of a variable of the class type

```
DigitalClock dc2 = new DigitalClock(12, 23);
```

Constructors

- Constructors can be overloaded as with any other method
- Each constructor in a given class must have a different signature (determined by the number, order and types of parameters)

Example

```
public class DigitalClock
{
   private int hours;
   private int minutes;

   public DigitalClock( )
   {    …
   }
   public DigitalClock(int h)
   {    …
   }
   public DigitalClock(int h, int m)
   {    …
   }
   …
```

Constructors

- What happens in those classes we have written that had no constructor defined?
- The compiler automatically creates a default constructor for any class that has no constructors defined by the programmer
- This automatically generated constructor allow us to create instances of the class, but has no other code

Examples - Default constructor

```
public class SimpleDigitalClock
{
	private int hours;
	private int minutes;
	// other methods ...

	public String toString()
	{
		return "SimpleDigitalClock[  hours: " + hours + ", minutes: " + minutes + " ]";
	}
}
public class SimpleDigitalClockTest
{
	public static void main(String [] args)
	{			DigitalClockSimple  dc = new DigitalClockSimple();
		System.out.println(  dc );
	}
}
```

Output

```
SimpleDigitalClock[  hours: 0, minutes: 0 ]
```

Constructors

- If you define one or more constructors, the compiler leaves all constructor definitions up to you and will not automatically generate any constructor
- If we define constructors and omit the definition of a default constructor, we cannot create a variable of that type with no parameters

```
DigitalClock dc = new DigitalClock( );
```

Constructors

- We can call one constructor from within another constructor of the same class
- A call to another constructor must be the first statement in the calling constructor

Example

```
public DigitalClock(int h)
{
    this(h, 0);
}
```

- Privacy leaks

Effects of multiple references

- We saw previously that when we have two references to a single object, changing the state of the object through either reference affects the one object

```
DigitalClock dc1 = new DigitalClock(12);
Digital dc2 = dc1;
dc1.setHours (13);
System.out.println(dc2);
```

Effects of multiple references

```
DigitalClock dc1 = new DigitalClock(12);
Digital dc2 = dc1;
dc1.setHours (13);
System.out.println(dc2);
```

Effects of multiple references

```
DigitalClock dc1 = new DigitalClock(12);
Digital dc2 = dc1;
dc1.setHours (13);
System.out.println(dc2);
```

Effects of multiple references

```
DigitalClock dc1 = new DigitalClock(12);
Digital dc2 = dc1;
dc1.setHours (13);
System.out.println(dc2);
```

Effects of multiple references

```
DigitalClock dc1 = new DigitalClock(12);
Digital dc2 = dc1;
dc1.setHours (13);
System.out.println(dc2);
```

Time[Hours = 13; Minutes = 0]

Effects of multiple references

- A similar situation arises when we return the value of an attribute as the result of a method
- If the returned value is of a primitive type, a copy of the value is returned
- If the returned value is of a class type, a copy of the reference is returned and the returned reference refers to the same object as the attribute

Privacy leaks

- The effects of multiple references can cause privacy leaks
- A privacy leak occurs when a private attribute of an object is accessible and can be changed from outside the object

Example

```
public class Appointment
{
    DigitalClock time;
    String date;
    String whoWith;
    String subject;

    public Appointment(int hour, int minute, String d, String who, String subj)
    {
        time = new DigitalClock( );
        time.setHours(hour);
        time.setMinutes(minute);
        date = d;
        whoWith = who;
        subject = subj;
    }
```

Example

```
    public DigitalClock getTime( )
    {
        return time;
    }

    public String getDate( )
    {
        return date;
    }

    public String toString( )
    {
        return "Appointment[\n  " + time + "\n  Date: " + date + "\n  WhoWith: " +
                    whoWith + "\n  Subject: " + subject + " ]";
    }
}
```

Example

```
public static void main(String[ ] args)
{
    Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
    System.out.println(a);

    DigitalClock dc = a.getTime( );
    dc.setHours(13);
    dc.setMinutes(15);

    String s = a.getDate( );
    s = "Tomorrow";

    System.out.println(a);
}
```

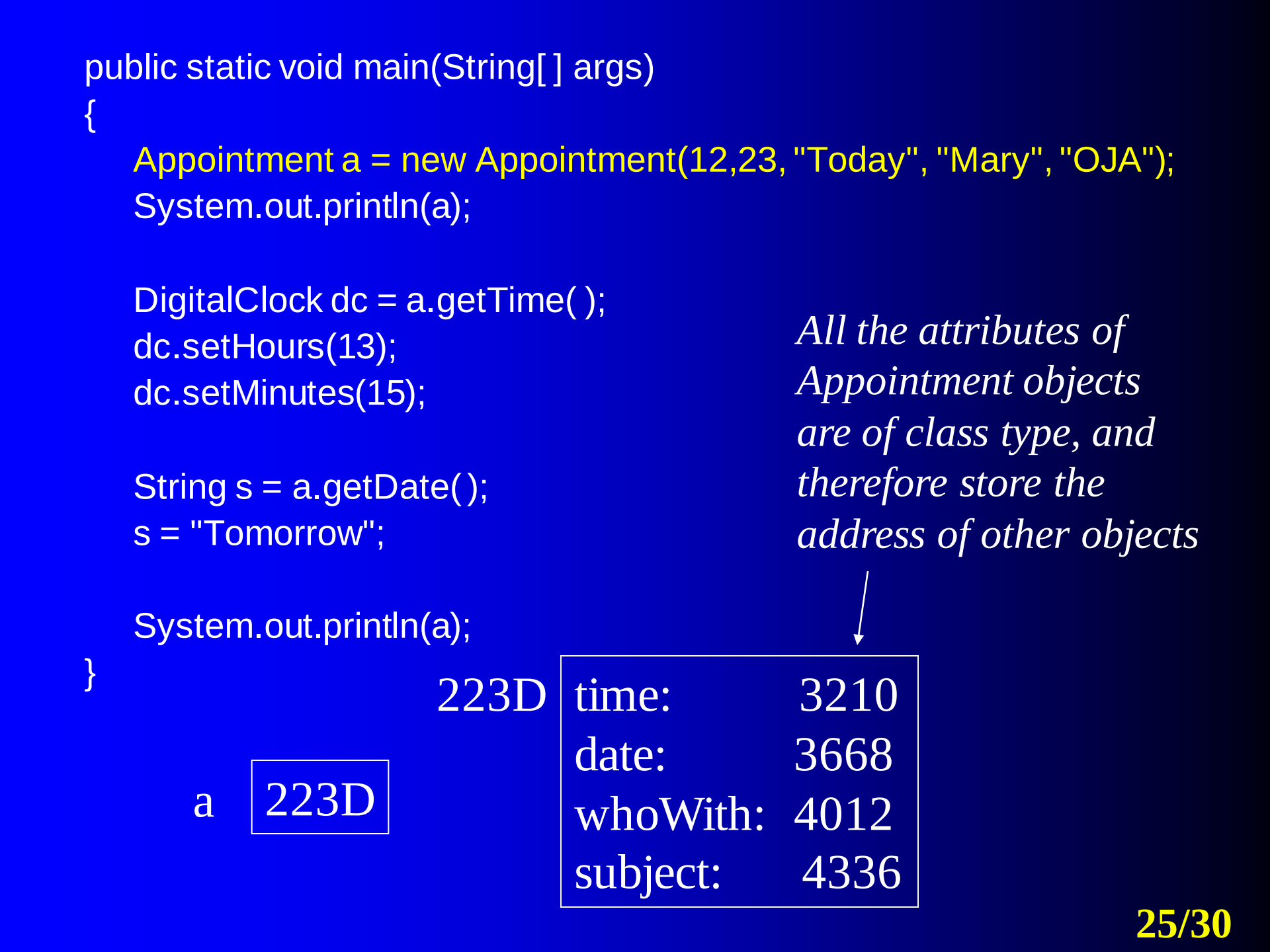
```
public static void main(String[ ] args)
{
    Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
    System.out.println(a);

    DigitalClock dc = a.getTime( );
    dc.setHours(13);
    dc.setMinutes(15);

    String s = a.getDate( );
    s = "Tomorrow";

    System.out.println(a);
}
```

All the attributes of Appointment objects are of class type, and therefore store the address of other objects

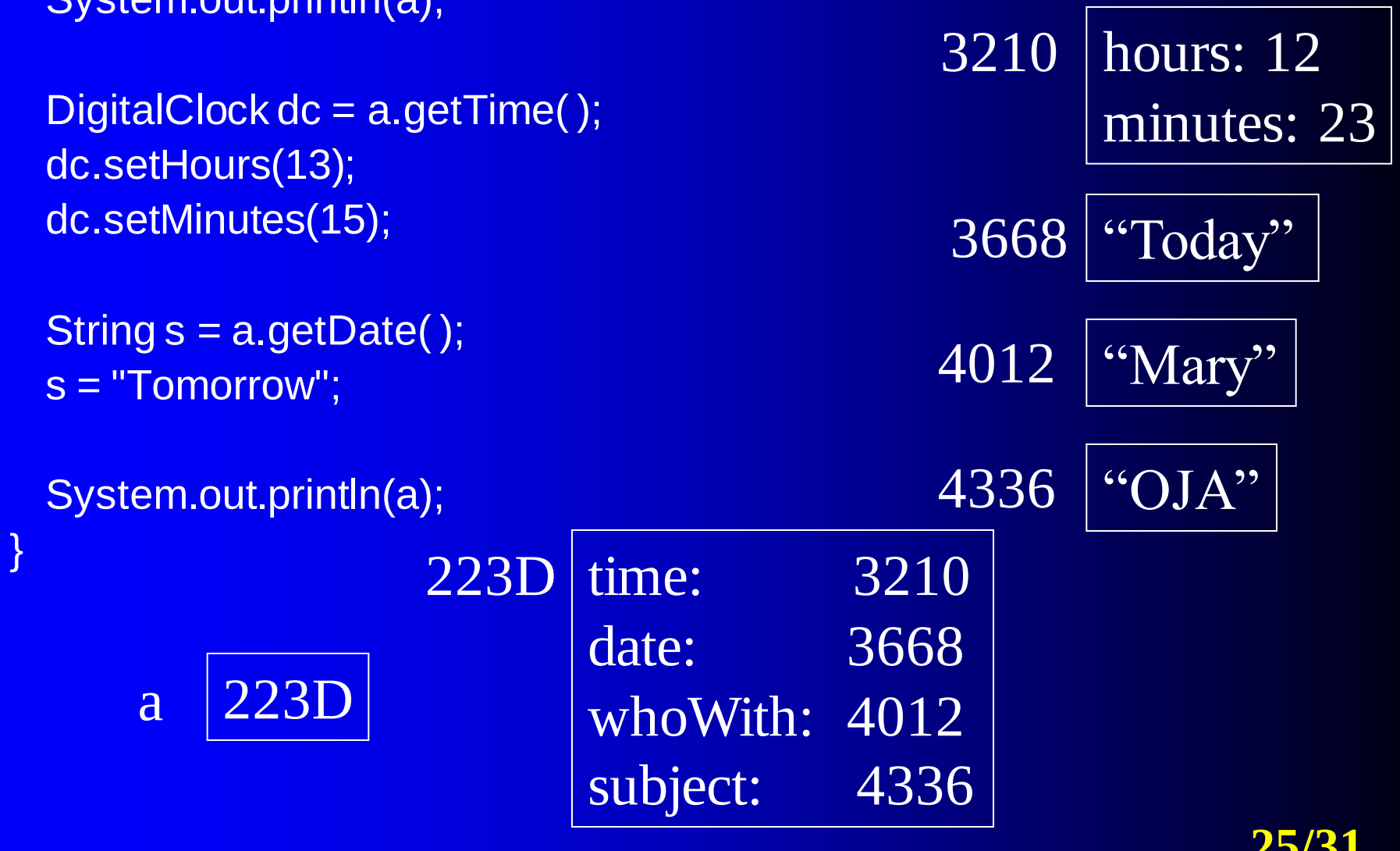


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```

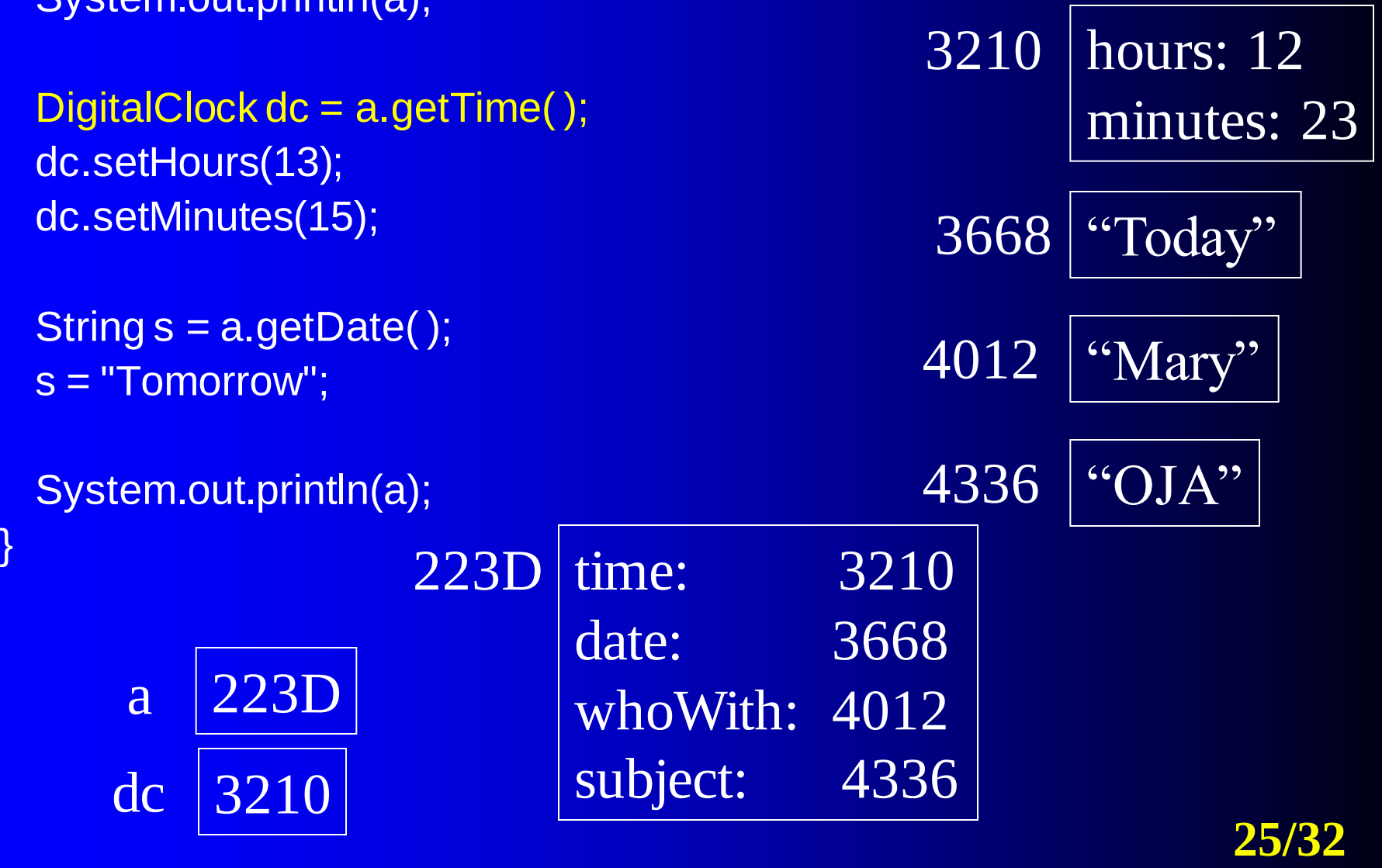


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```

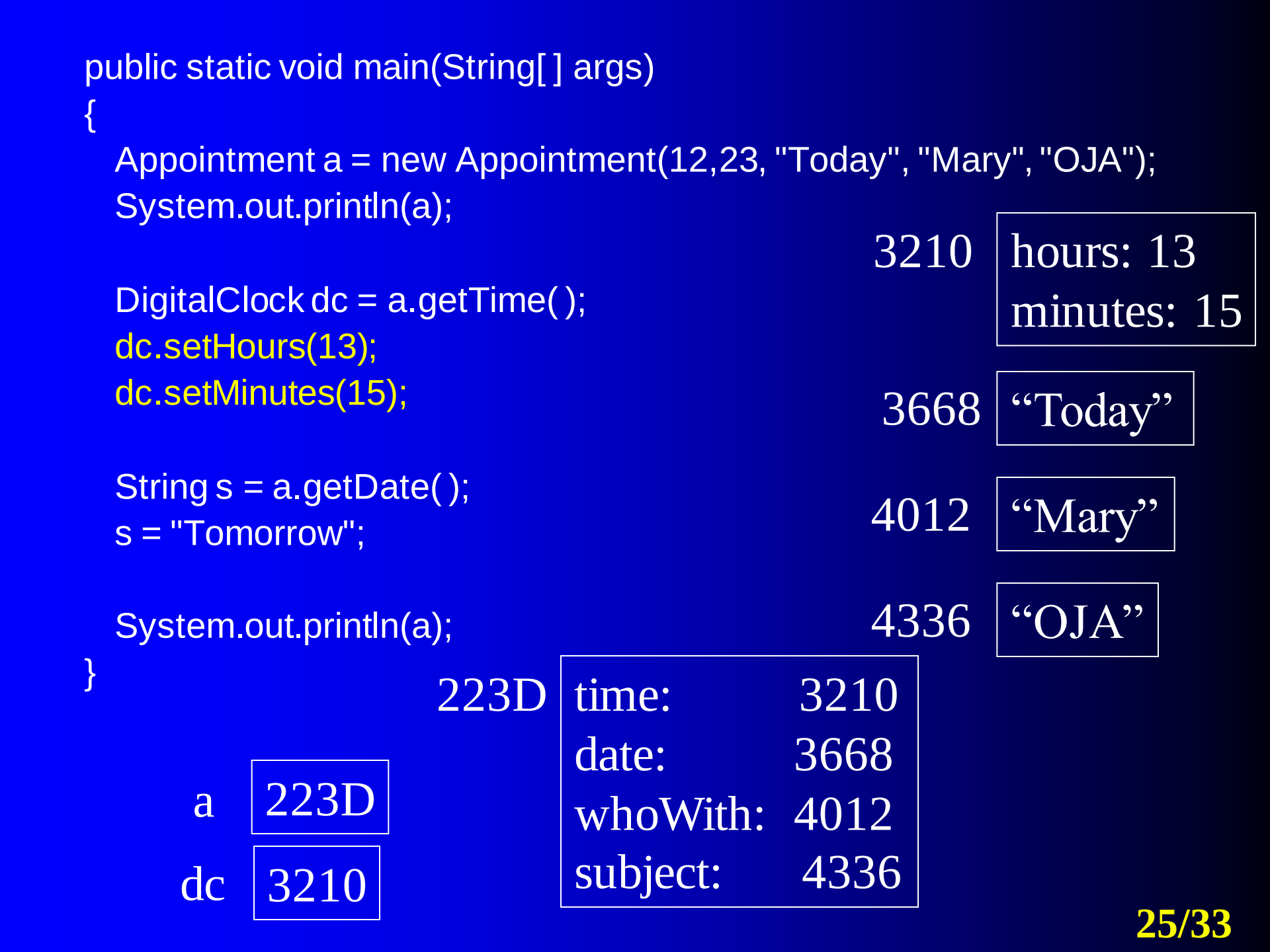


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```

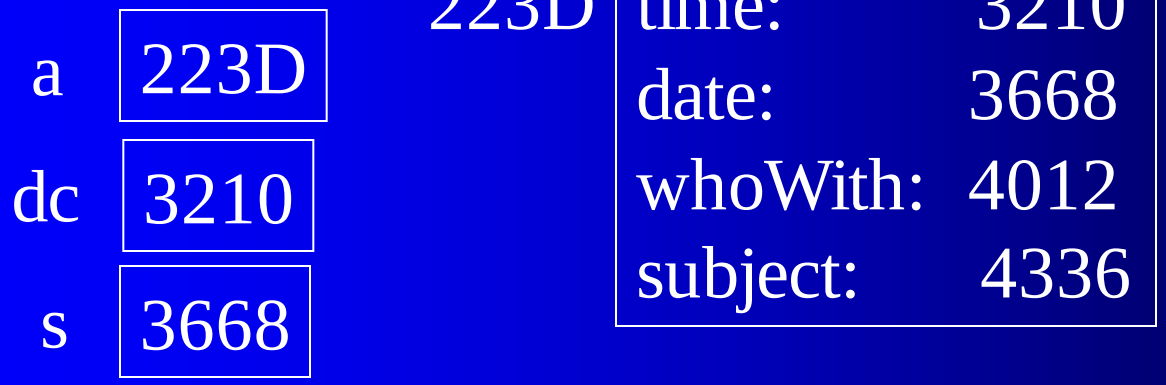


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```

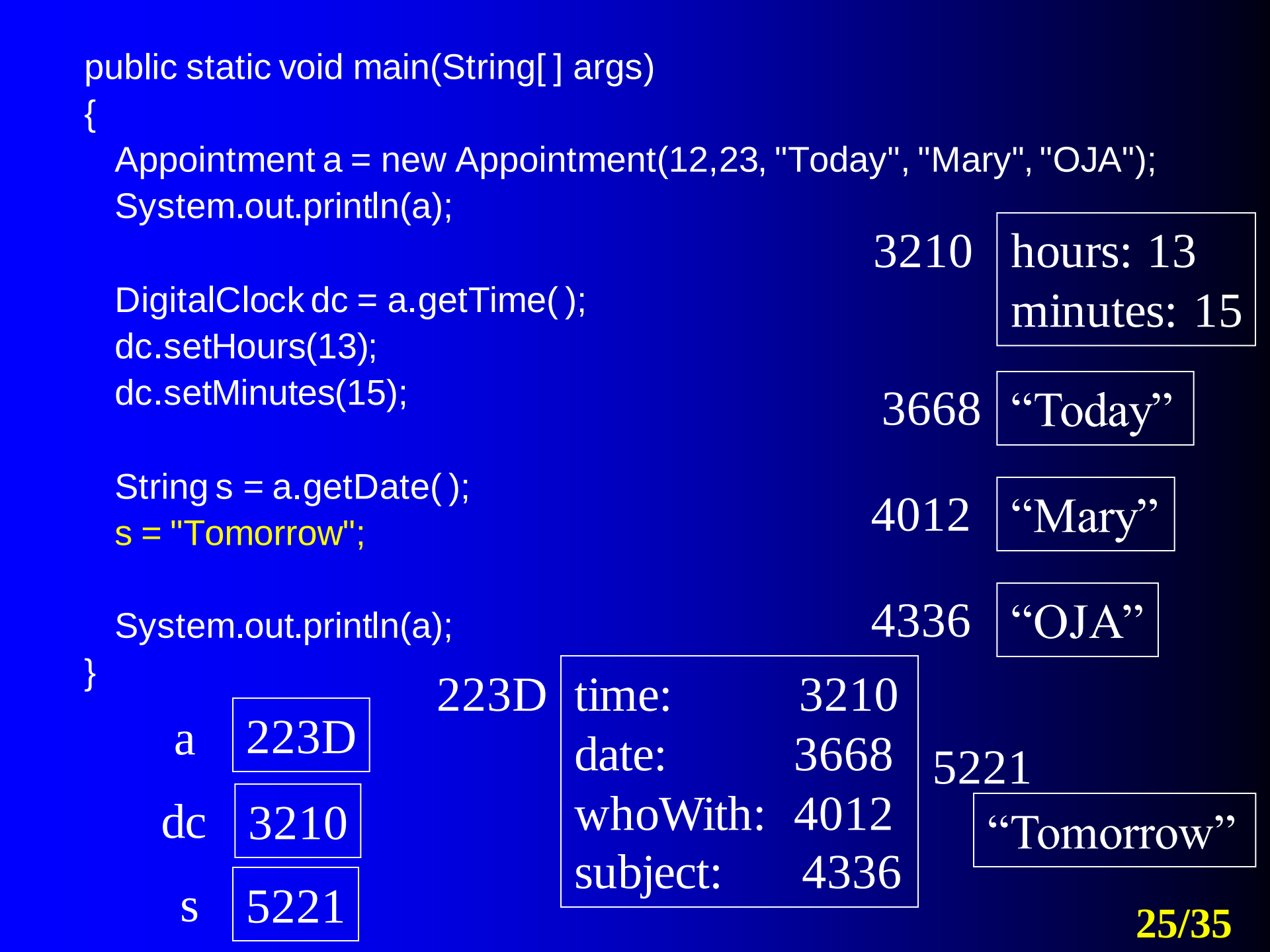


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```



Result

Appointment[
  Time[hours: 12, minutes: 23]
  Date: Today
  WhoWith: Mary
  Subject: OJA]
Appointment[
  Time[hours: 13, minutes: 15]
  Date: Today
  WhoWith: Mary
  Subject: OJA]

Preventing privacy leaks

- Returning an object that can be altered results in a privacy leak
- It is *safe* to return a reference with Strings as the String object is immutable (cannot be changed)

Preventing privacy leaks by cloning

- Instead of returning a reference to a mutable attribute, we need to return a clone (exact copy) of the object
- We may do this by defining a clone method in the class of the attribute's type

Example

```
public class Appointment
{
    …
    public DigitalClock getTime( )
    {
        return (DigitalClock) time.clone( );
    }
    …
}
```

Example

```
public class DigitalClock
{
    …
    public Object clone( )
    {
        DigitalClock cloned = new DigitalClock(hours, minutes);
        return cloned;
    }
    …
}
```

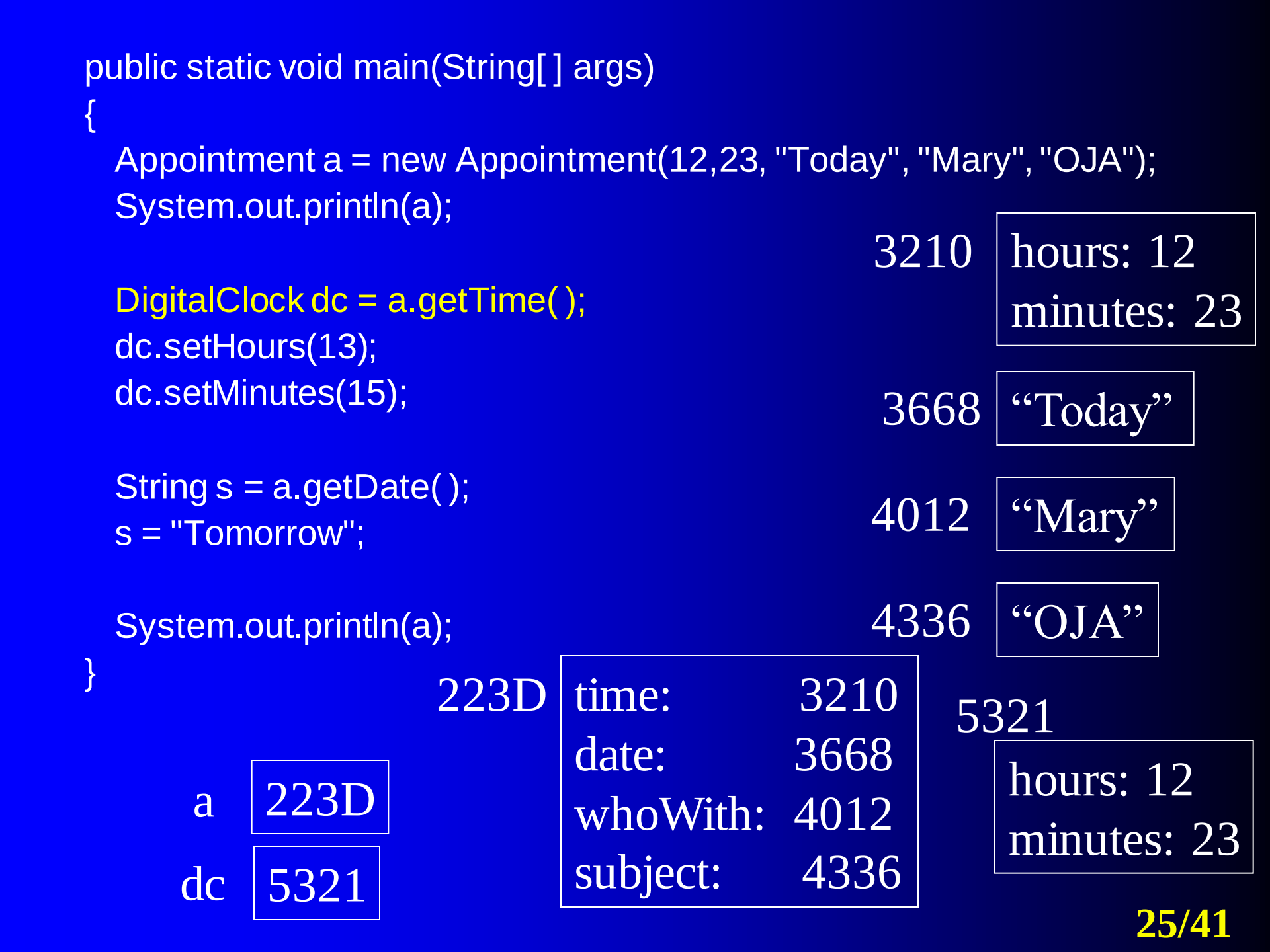
** This is a minimal explanation of cloning!*

```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```

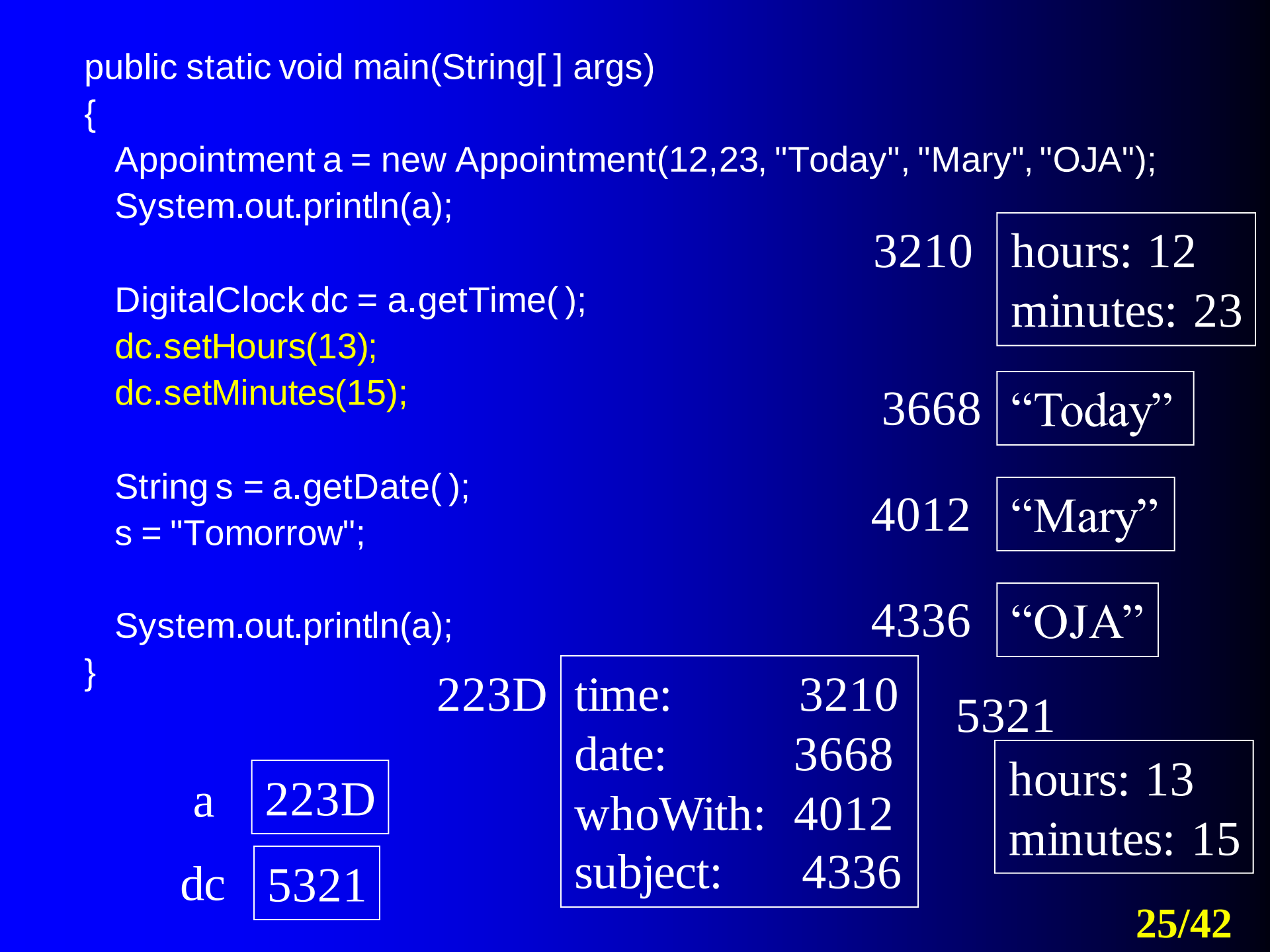


```
public static void main(String[ ] args)
{
   Appointment a = new Appointment(12,23, "Today", "Mary", "OJA");
   System.out.println(a);

   DigitalClock dc = a.getTime( );
   dc.setHours(13);
   dc.setMinutes(15);

   String s = a.getDate( );
   s = "Tomorrow";

   System.out.println(a);
}
```



Example

Appointment[
  Time[hours: 12, minutes: 23]
  Date: Today
  WhoWith: Mary
  Subject: OJA]
Appointment[
  Time[hours: 12, minutes: 23]
  Date: Today
  WhoWith: Mary
  Subject: OJA]

- Using packages

Packages

- Packages are a way of grouping classes so that they can be used in other programs without placing a copy of them in the directory with the new program
  - Collection of classes
  - Grouped in a directory
  - Given a package name

Packages

- At the start of each file in the package write

  ```
  package packageName;
  ```

- Any class outside the package using one of the classes in the package would have to include an import statement

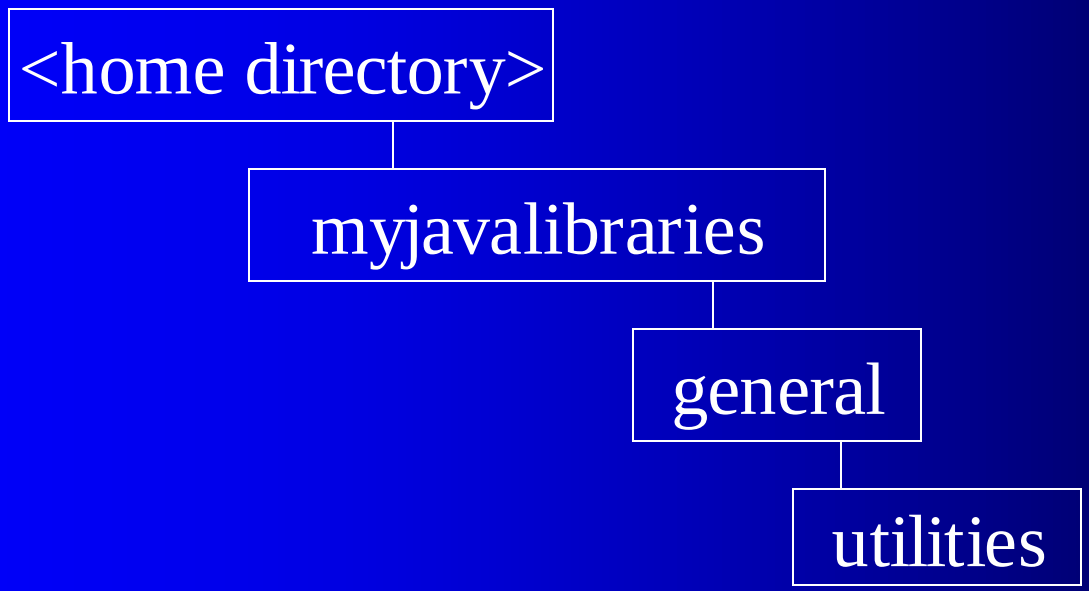
  ```
  import packageName;
  ```

Package names

- The import statement tells the compiler where to find the classes in the package
- To find a package Java needs
  - The name of the package
  - The classpath variable

Package names

- The package name is based on the name of the directory in which it is located
- For example, given the following directory structure in my home directory



Packages

- If I place the DigitalClock class in the utilities directory, the package name would be general.utilities
- At the top of the DigitalClock.java file, I would have to place the package statement
  - package general.utilities;
- The directory myjavalibraries would have to be added to the CLASSPATH environment variable so that Java would know to look for packages there

Packages

- Change your CLASSPATH variable in your .cshrc file

```
setenv CLASSPATH ~myUserName/myjavalibraries:.
```

- Separate classpath names in Unix with a :
- Make sure you include the current directory (.) in your class path environment variable

Packages

- If the Appointment class is in a different directory (and a different package) to the DigitalClock class, at the top of the file Appointment.java include the command

```
import general.utilities.DigitalClock;
```

or

```
import general.utilities.*;
```

Next lecture

- Array basics