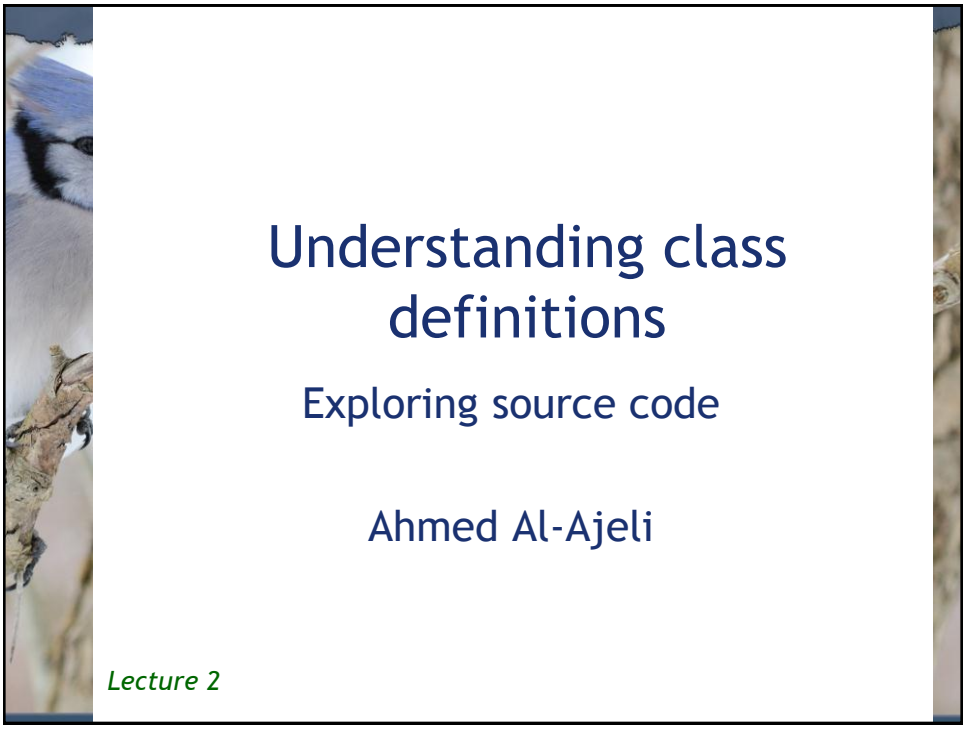
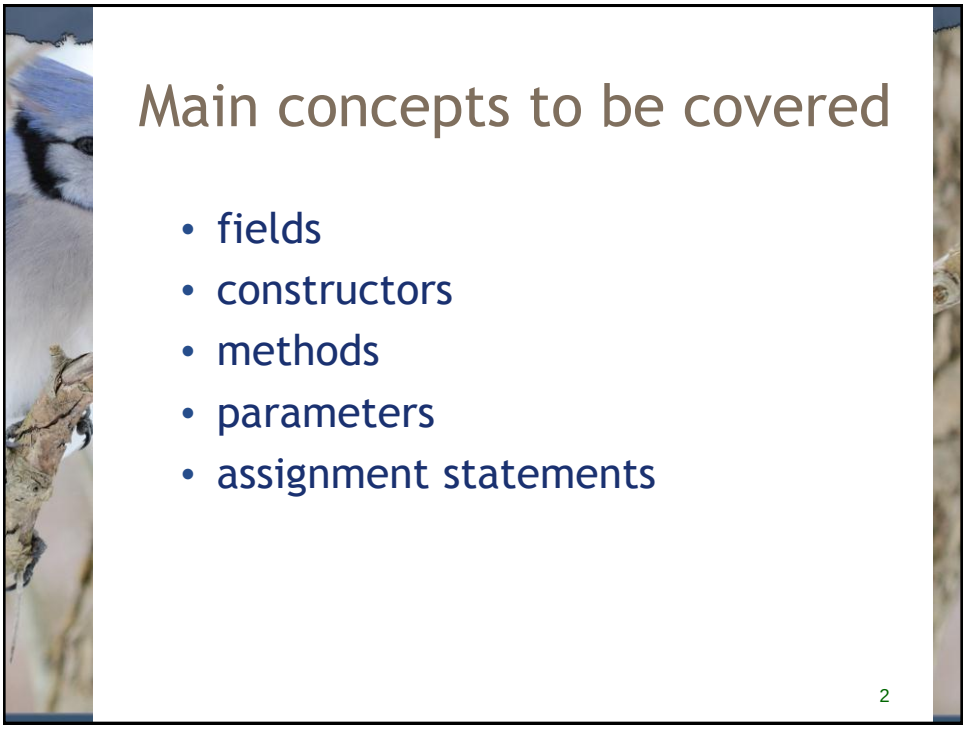
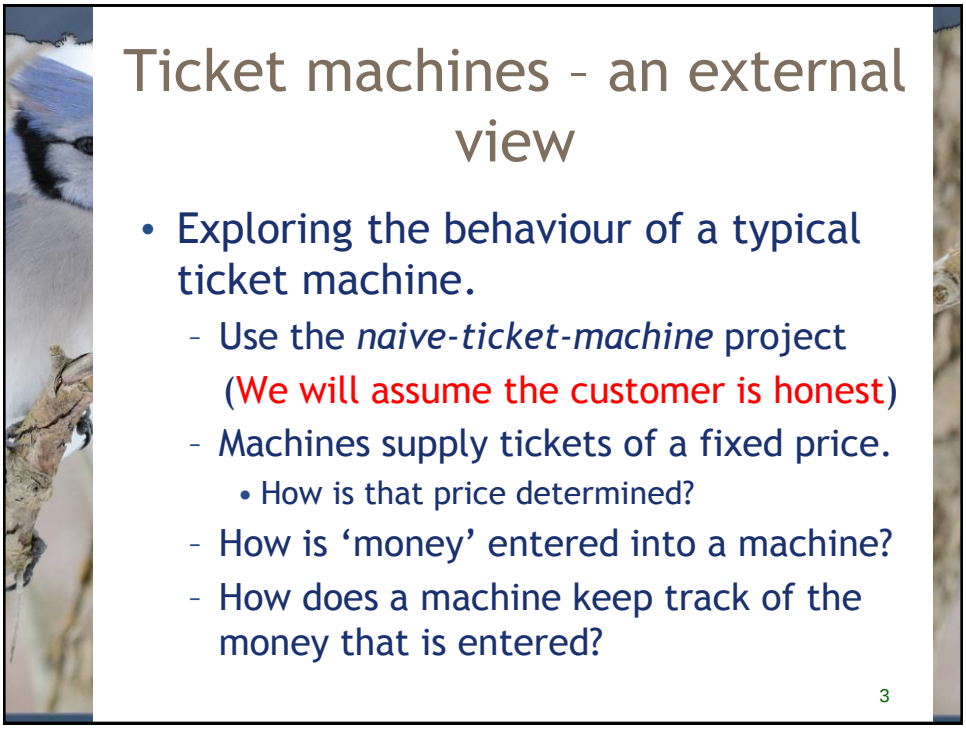
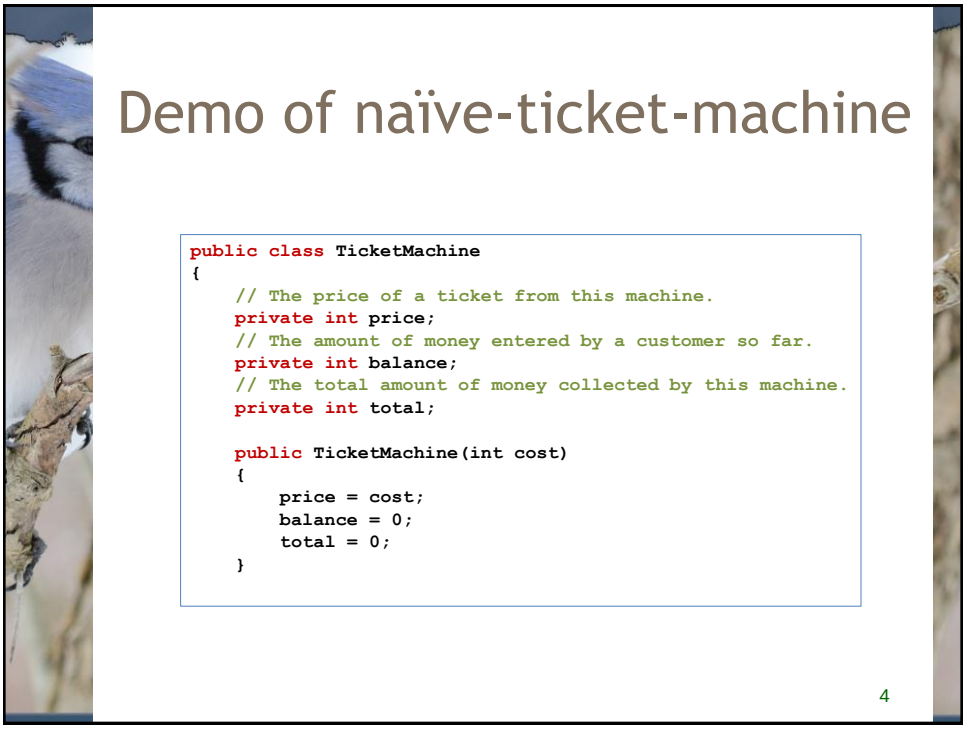
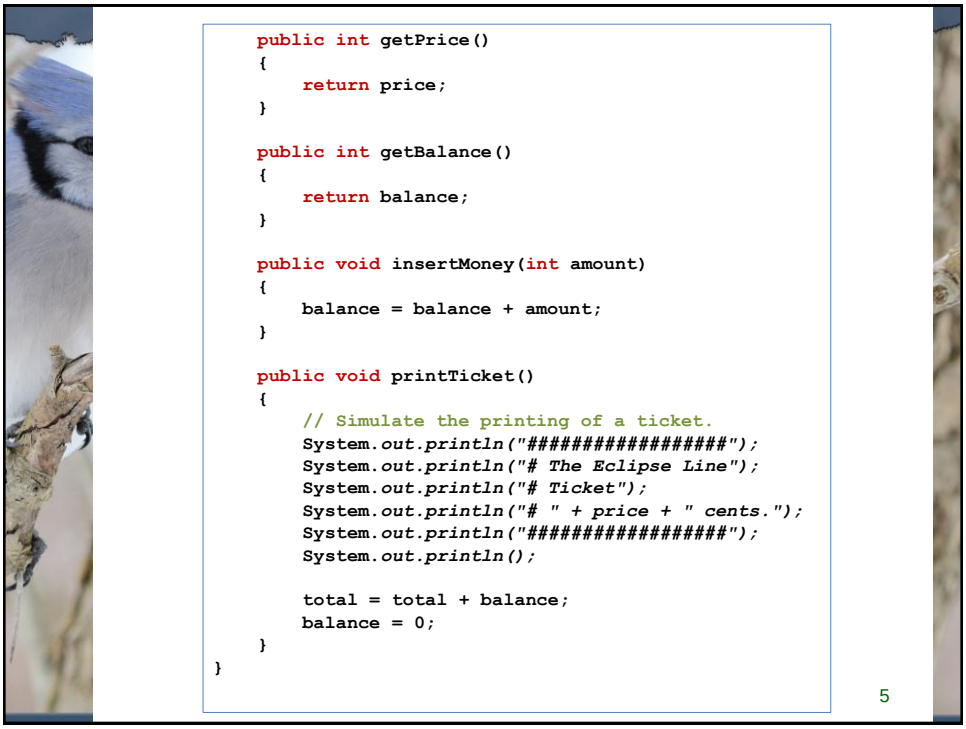

Understanding class definitions
Exploring source code
Ahmed Al-Ajeli
Lecture 2


Main concepts to be covered
• fields
• constructors
• methods
• parameters
• assignment statements
2

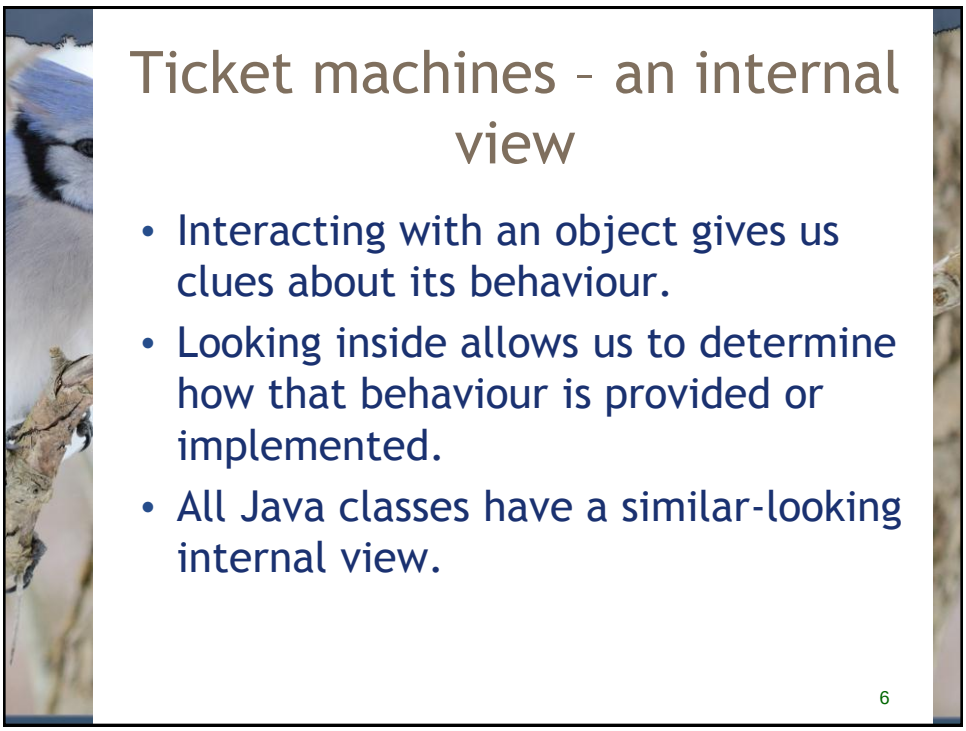

Ticket machines – an external view
• Exploring the behaviour of a typical ticket machine.
– Use the *naive-ticket-machine* project
(We will assume the customer is honest)
– Machines supply tickets of a fixed price.
• How is that price determined?
– How is 'money' entered into a machine?
– How does a machine keep track of the money that is entered?
3


Demo of naïve-ticket-machine
public class TicketMachine
{
// The price of a ticket from this machine.
private int price;
// The amount of money entered by a customer so far.
private int balance;
// The total amount of money collected by this machine.
private int total;
public TicketMachine(int cost)
{
price = cost;
balance = 0;
total = 0;
}
4



```
    public int getPrice()
    {
        return price;
    }

    public int getBalance()
    {
        return balance;
    }

    public void insertMoney(int amount)
    {
        balance = balance + amount;
    }

    public void printTicket()
    {
        // Simulate the printing of a ticket.
        System.out.println("###################");
        System.out.println("# The Eclipse Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("###################");
        System.out.println();

        total = total + balance;
        balance = 0;
    }
}
```

5



Ticket machines – an internal view

- Interacting with an object gives us clues about its behaviour.
- Looking inside allows us to determine how that behaviour is provided or implemented.
- All Java classes have a similar-looking internal view.

6

Basic class structure

```
public class TicketMachine
{
    Inner part omitted.
}

public class ClassName
{
    Fields
    Constructors
    Methods
}
```

The outer wrapper of TicketMachine

The inner contents of a class

7

Keywords

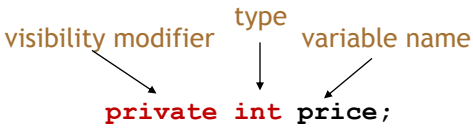
- Words with a special meaning in the language:
  - **public**
  - **class**
  - **private**
  - **int**
- Also known as *reserved words*.
- Always entirely lower-case.

8

Fields

- Fields store values for an object.
- They are also known as *instance variables*.
- Fields define the state of an object.
- Some values change often.
- Some change rarely (or not at all).

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    Further details omitted.
}
```



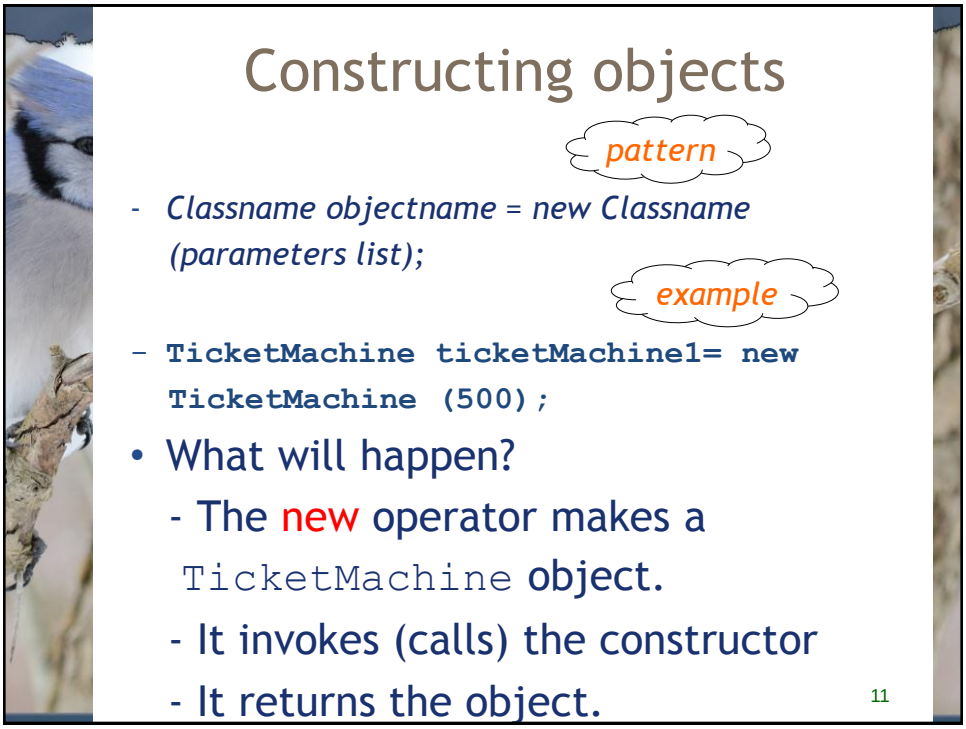
9

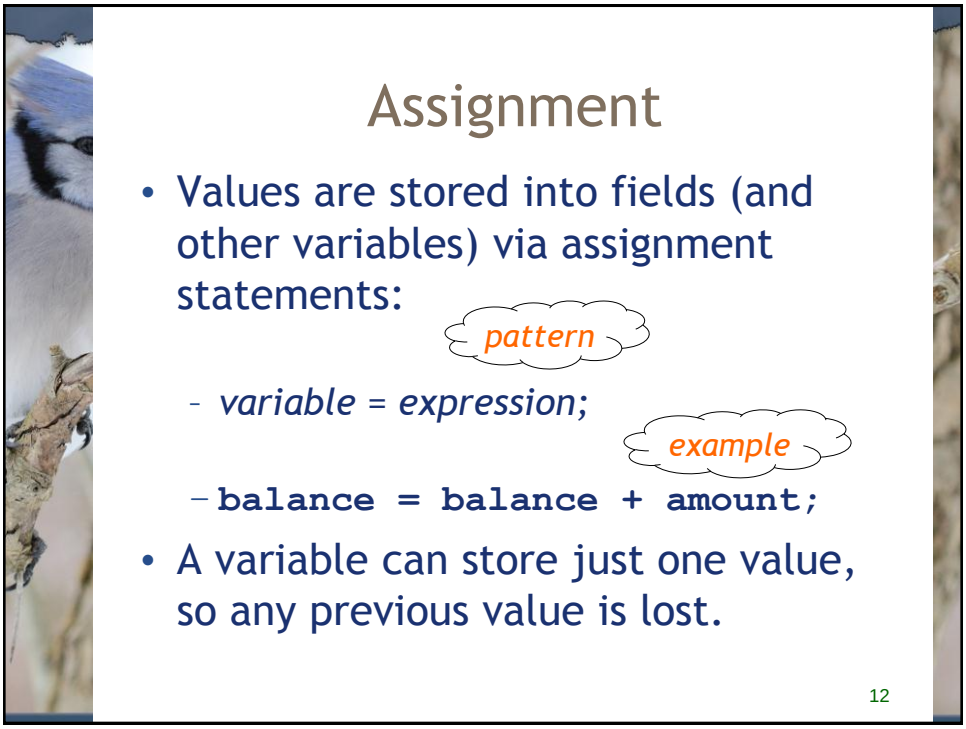
Constructors

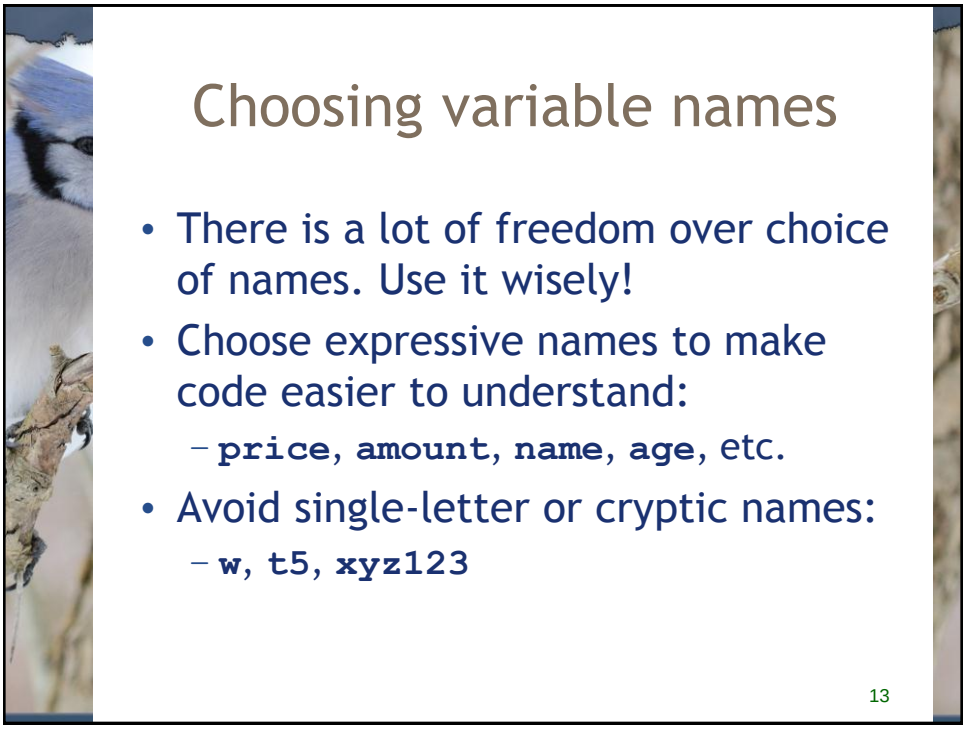
```
public TicketMachine(int cost)
{
    price = cost;
    balance = 0;
    total = 0;
}
```

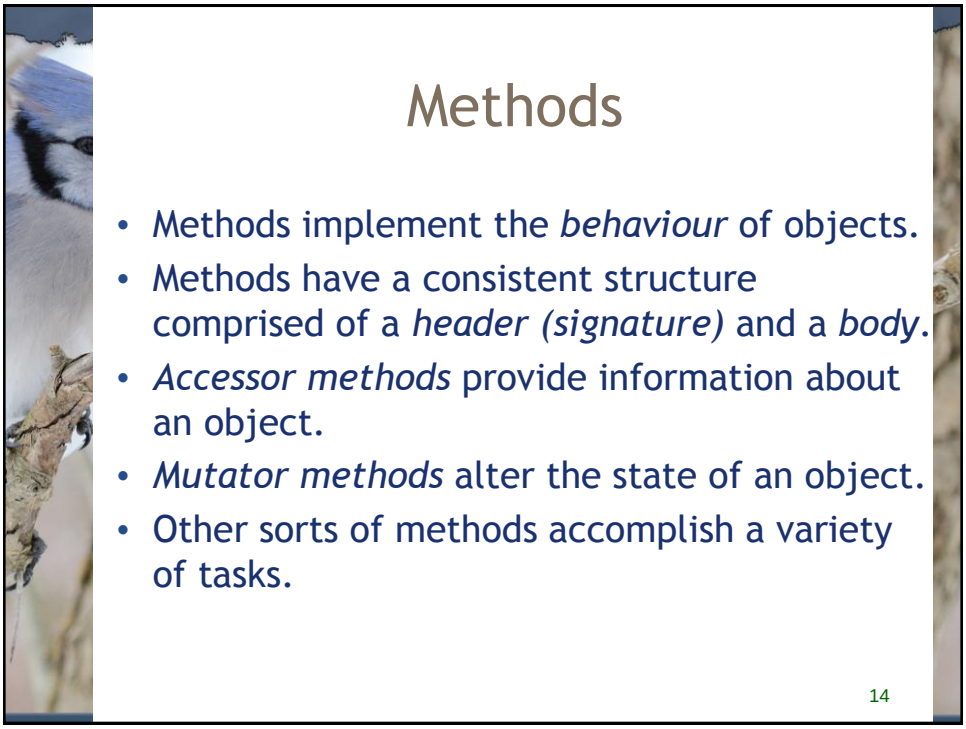
- Initialize an object.
- Have the same name as their class.
- Public visibility (since called by the user)
- Close association with the fields:
  - Initial values stored into the fields.
  - Parameter values often used for these.

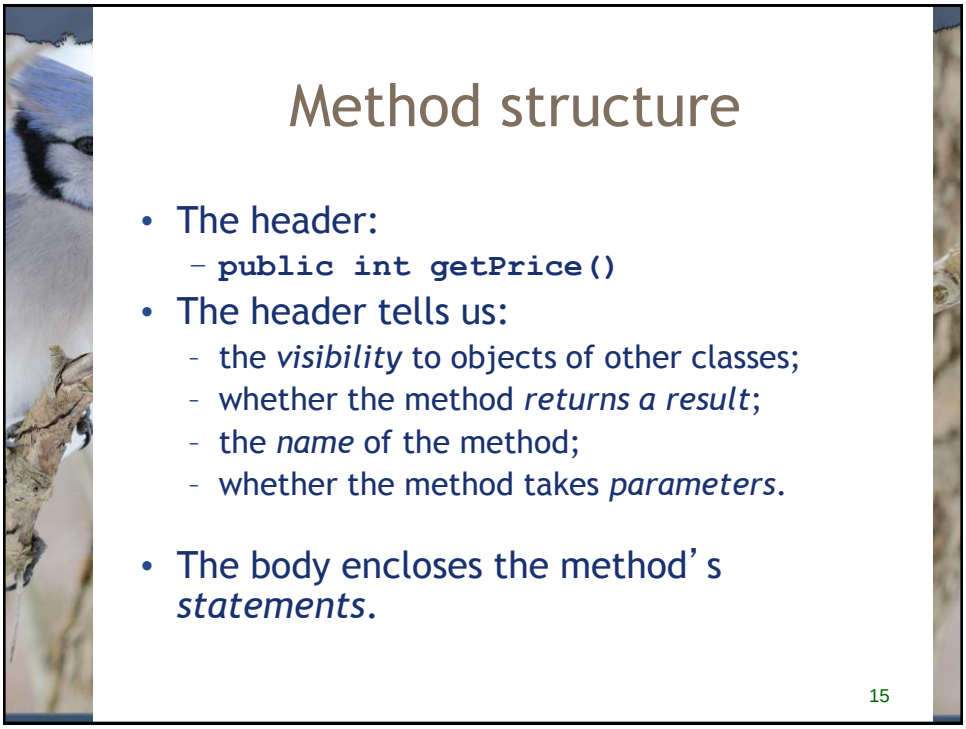
10

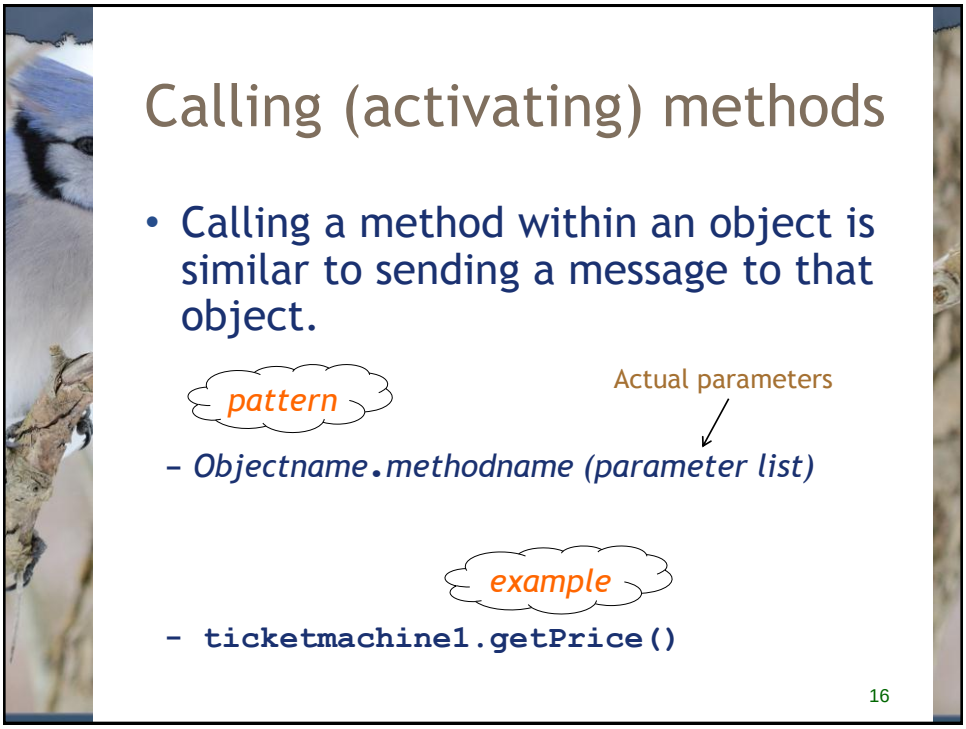

Constructing objects
pattern
- Classname objectname = new Classname (parameters list);
example
- TicketMachine ticketMachine1= new TicketMachine (500);
• What will happen?
- The new operator makes a TicketMachine object.
- It invokes (calls) the constructor
- It returns the object.
11


Assignment
• Values are stored into fields (and other variables) via assignment statements:
pattern
- variable = expression;
example
- balance = balance + amount;
• A variable can store just one value, so any previous value is lost.
12

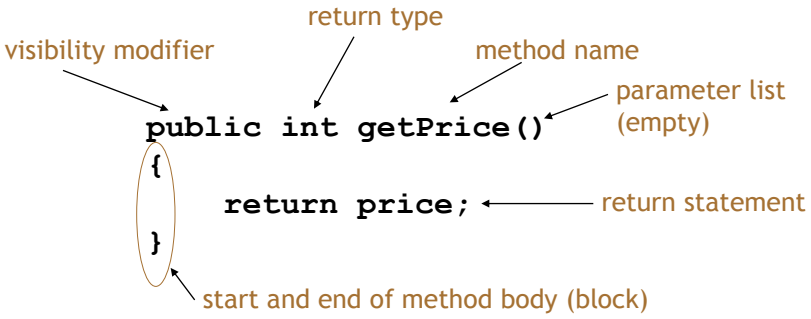

Choosing variable names
There is a lot of freedom over choice of names. Use it wisely!
Choose expressive names to make code easier to understand:
price, amount, name, age, etc.
Avoid single-letter or cryptic names:
w, t5, xyz123
13


Methods
Methods implement the behaviour of objects.
Methods have a consistent structure comprised of a header (signature) and a body.
Accessor methods provide information about an object.
Mutator methods alter the state of an object.
Other sorts of methods accomplish a variety of tasks.
14


Method structure
• The header:
– public int getPrice()
• The header tells us:
– the visibility to objects of other classes;
– whether the method returns a result;
– the name of the method;
– whether the method takes parameters.
• The body encloses the method's statements.
15


Calling (activating) methods
• Calling a method within an object is similar to sending a message to that object.
pattern
Actual parameters
– Objectname.methodname (parameter list)
example
– ticketmachine1.getPrice()
16

Accessor (`get`) methods

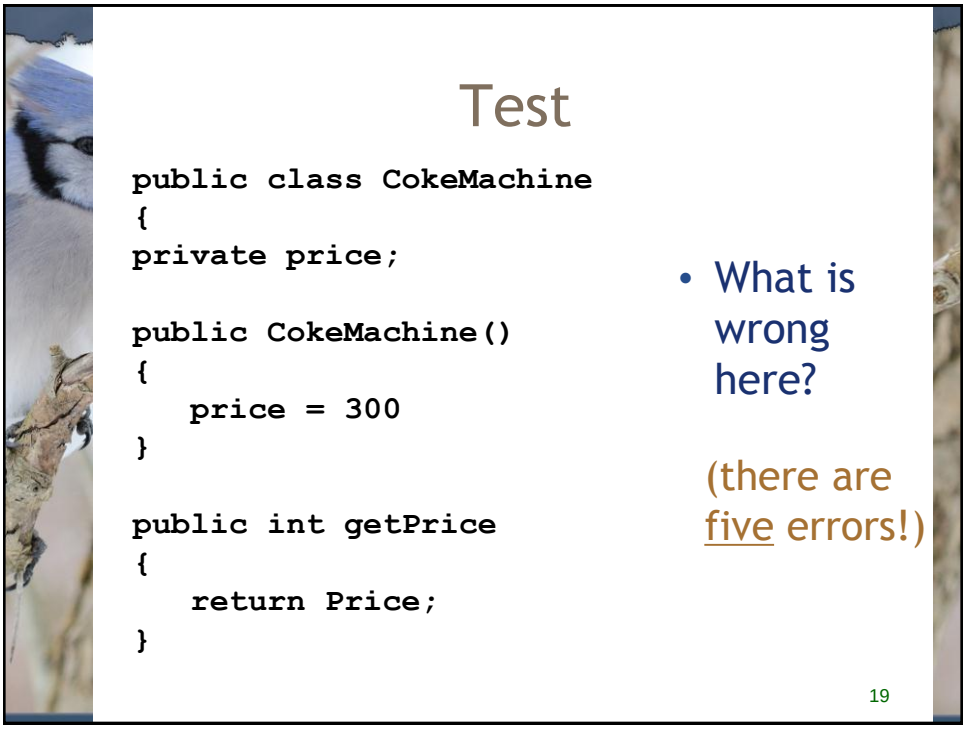


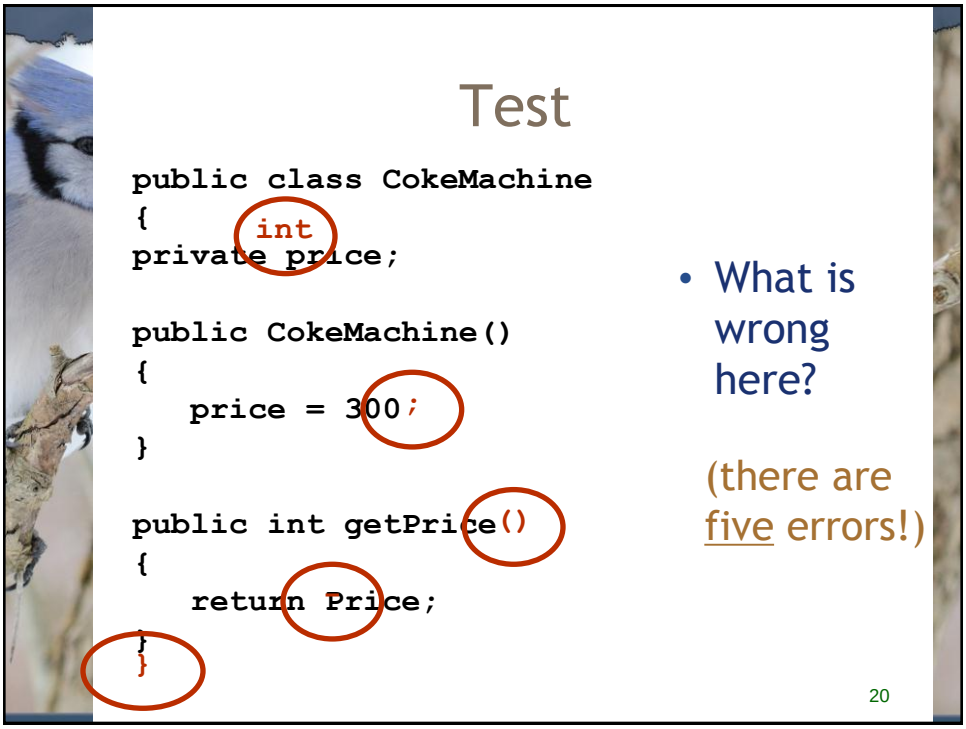
17

Accessor methods

- An accessor method always has a return type that is not **`void`**.
- An accessor method returns a value (*result*) of the type given in the header.
- The method will contain a **`return`** statement to return the value.
- NB: Returning is *not* printing!

18


Test
public class CokeMachine
{
private price;
public CokeMachine()
{
price = 300
}
public int getPrice
{
return Price;
}
• What is wrong here?
(there are five errors!)
19

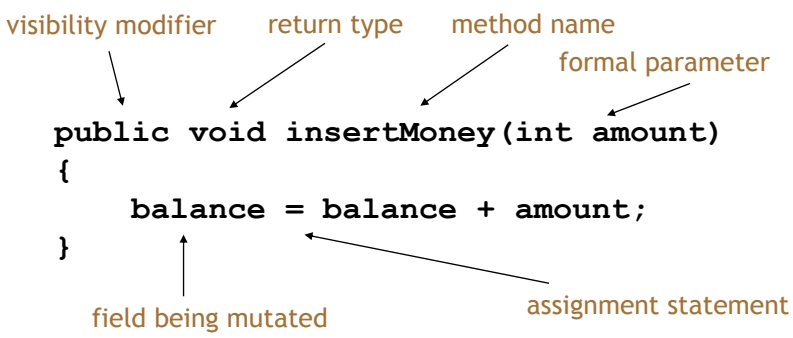

Test
public class CokeMachine
{
int
private price;
public CokeMachine()
{
price = 300;
}
public int getPrice()
{
return Price;
}
}
• What is wrong here?
(there are five errors!)
20

Mutator methods

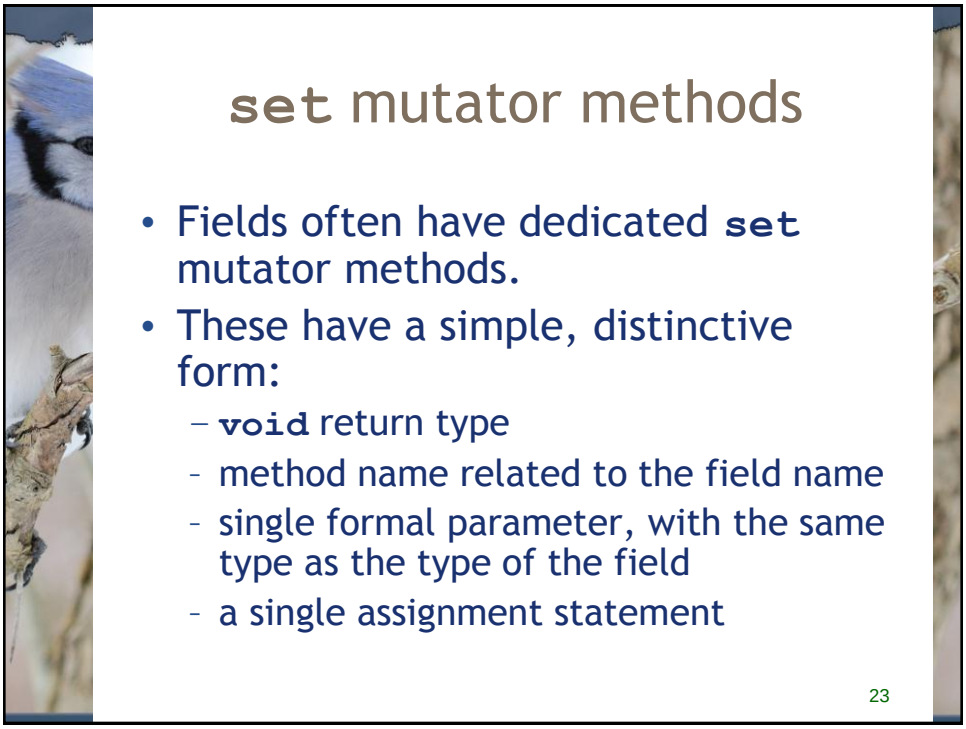
- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
  - They typically contain one or more assignment statements.
  - Often receive parameters.

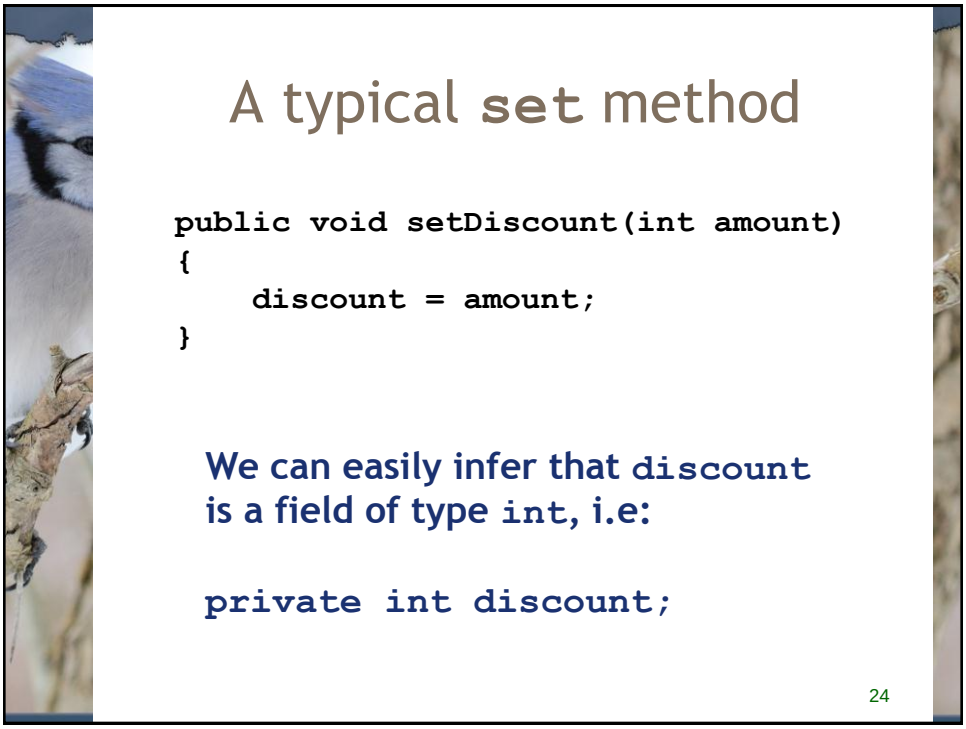
21

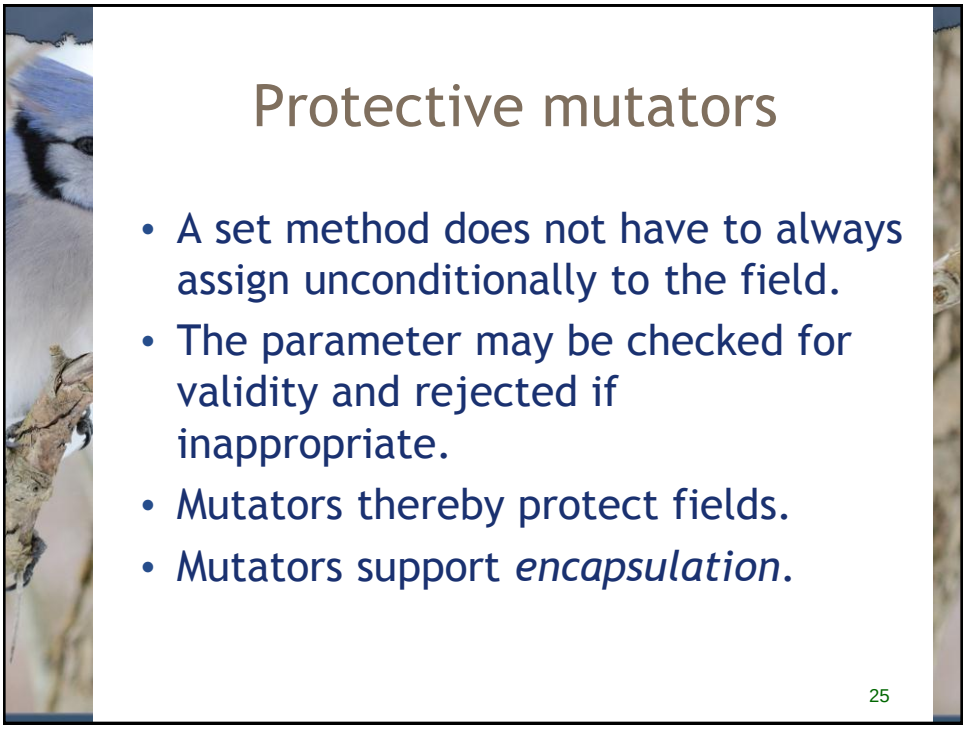
Mutator methods

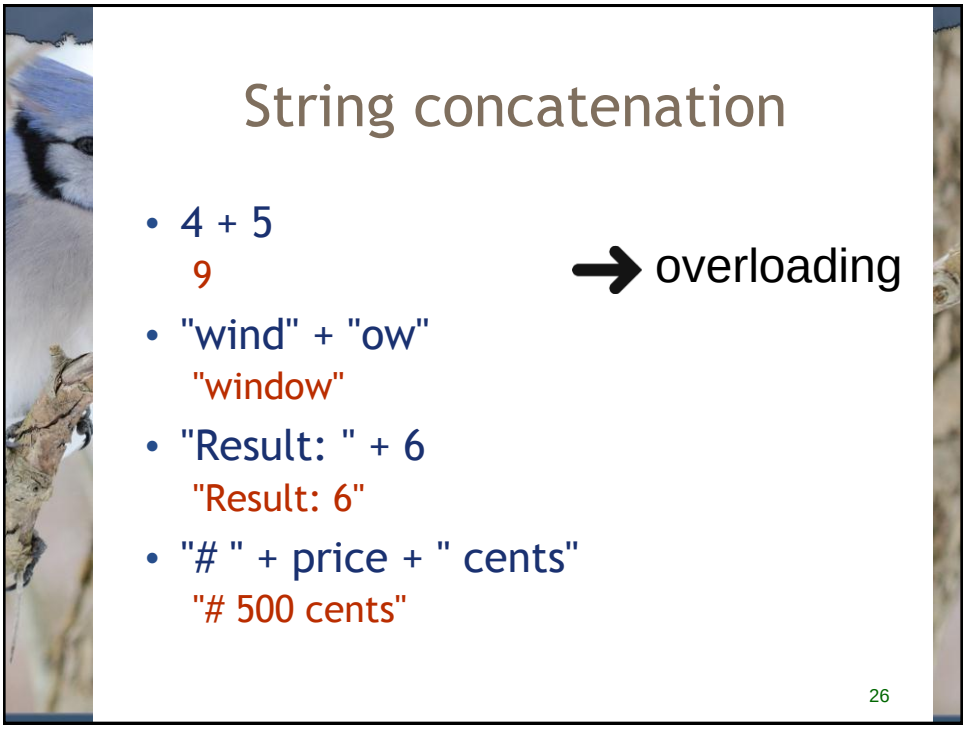


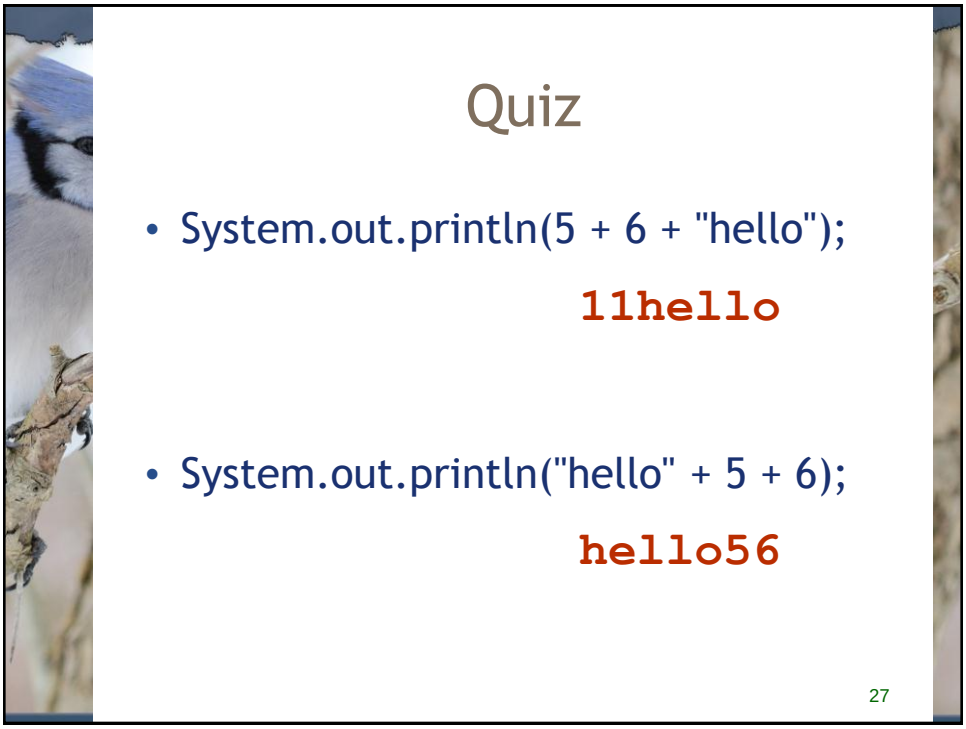
22


set mutator methods
• Fields often have dedicated set mutator methods.
• These have a simple, distinctive form:
– void return type
– method name related to the field name
– single formal parameter, with the same type as the type of the field
– a single assignment statement
23


A typical set method
public void setDiscount(int amount)
{
discount = amount;
}
We can easily infer that discount is a field of type int, i.e:
private int discount;
24


Protective mutators
• A set method does not have to always assign unconditionally to the field.
• The parameter may be checked for validity and rejected if inappropriate.
• Mutators thereby protect fields.
• Mutators support *encapsulation*.
25


String concatenation
• 4 + 5
9
→ overloading
• "wind" + "ow"
"window"
• "Result: " + 6
"Result: 6"
• "# " + price + " cents"
"# 500 cents"
26


Quiz
• System.out.println(5 + 6 + "hello");
11hello
• System.out.println("hello" + 5 + 6);
hello56
27