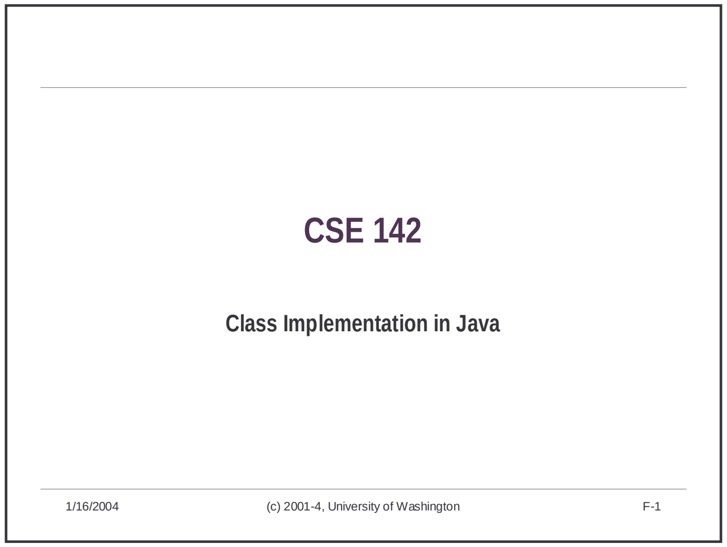
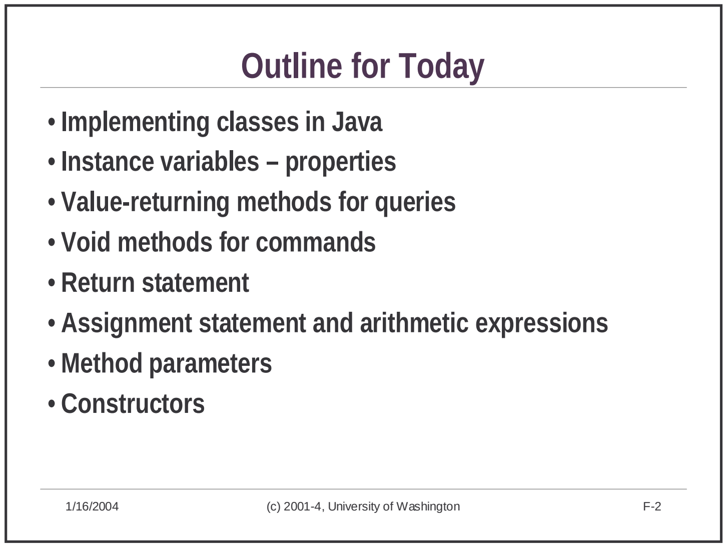


CSE 142

Class Implementation in Java

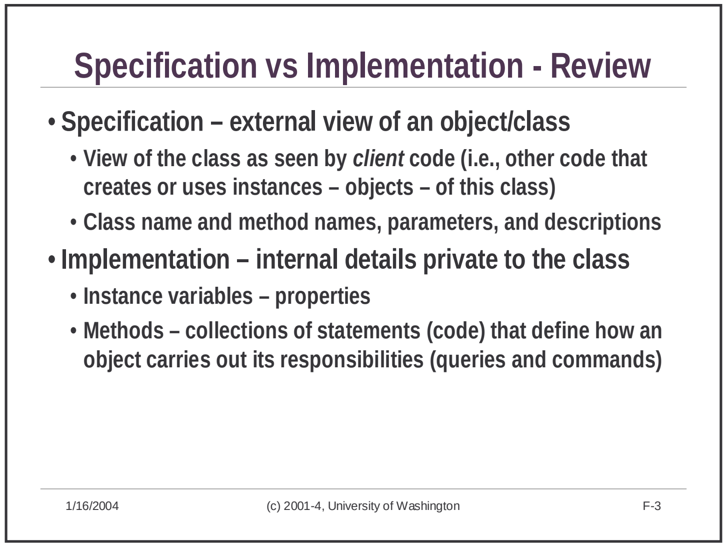
1/16/2004 (c) 2001-4, University of Washington F-1



Outline for Today

- Implementing classes in Java
- Instance variables – properties
- Value-returning methods for queries
- Void methods for commands
- Return statement
- Assignment statement and arithmetic expressions
- Method parameters
- Constructors

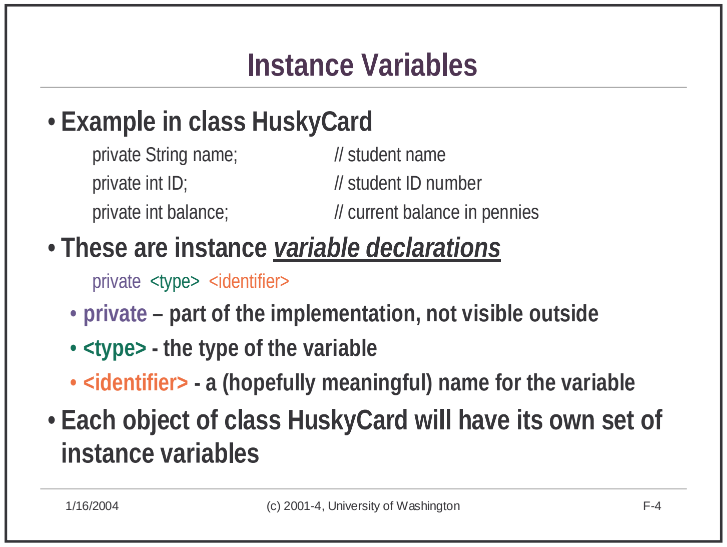
1/16/2004 (c) 2001-4, University of Washington F-2



Specification vs Implementation - Review

- Specification – external view of an object/class
  - View of the class as seen by *client* code (i.e., other code that creates or uses instances – objects – of this class)
  - Class name and method names, parameters, and descriptions
- Implementation – internal details private to the class
  - Instance variables – properties
  - Methods – collections of statements (code) that define how an object carries out its responsibilities (queries and commands)

1/16/2004 (c) 2001-4, University of Washington F-3



Instance Variables

- Example in class HuskyCard

```
private String name;          // student name
private int ID;               // student ID number
private int balance;          // current balance in pennies
```

- These are instance ***variable declarations***

```
private <type> <identifier>
```

  - private – part of the implementation, not visible outside
  - <type> - the type of the variable
  - <identifier> - a (hopefully meaningful) name for the variable
- Each object of class HuskyCard will have its own set of instance variables

1/16/2004 (c) 2001-4, University of Washington F-4

Constructors

- Whenever an object (instance of a class) is created, a *constructor* is executed
  - Idea: The constructor implementation should initialize the state of the object to some appropriate value(s)
- Like a command but named the same as the class
- Specification for HuskyCard

```
/** Construct a new HuskyCard with an initial balance of 0
  *  @param studentName  the student's name
  *  @param IDNumber the student's ID Number */
  public HuskyCard(String studentName, int IDNumber) { … }
```

  - This constructor has two *parameters* – studentName and IDNumber

1/16/2004 (c) 2001-4, University of Washington F-5

Constructor Implementation

- Idea: use parameter values as initial values for new object's state

```
/** Construct a new HuskyCard with an initial balance of 0
  *  @param studentName  the student's name
  *  @param IDNumber the student's ID Number */
public HuskyCard(String studentName, int IDNumber) {
    name = studentName;
    ID = IDnumber;
    balance = 0;
}
```

1/16/2004 (c) 2001-4, University of Washington F-6

Assignment Statement

- First example of a *statement*
- Syntax

  variable = *expression* ;
- Meaning
  - First, evaluate the expression (formula) to get a value
  - Second, bind that value to the *variable* whose name appears on the left
  - These two steps are done in that order, not simultaneously
  - Question: what does this mean (or do)?

    count = count + 1;

1/16/2004 (c) 2001-4, University of Washington F-7

Arithmetic Expressions

- Basic components
  - Literals – 17, 3.0, 1.023e23
  - Variable names – value is the current value of the variable
- Operators (see book for all the details)
  - +, -, *, /, % (remainder)

    Gotchas: for ints, x/y yields integer part, dropping any fraction; x%y gives the remainder
  - Operators have the usual ***precedence***

    For example, a + b * c is understood to mean a + (b * c)
  - Binary operators (ones that have two components) are ***left associative*** : a * b / c means (a * b) / c

    Use parentheses where needed to override or clarify: a * (b / c)
  - Mixing ints and doubles is normally ok – the int is converted to a double and the calculation is done as a double

1/16/2004 (c) 2001-4, University of Washington F-8

Implementing Methods for Simple Queries

- Example in class HuskyCard

```
/** return the name associated with this HuskyCard
 * @return this HuskyCard owner's name */
public String getName( ) {
    return name;
}
```

- When this method is executed, it replies with the value of the instance variable name

Test

- Let's try it out!
- Step 1: click the "compile" button to translate the code from text to something the Java machine can execute
- Step 2: enter commands in DrJava's interactions window to create an object and call one of its methods

```
HuskyCard card = new HuskyCard("E. Fudd", 1020304);
card.getName()
```

More About Value-Returning (Query) Methods

- Form

```
/** Comment specifying the method */
public <result type> <identifier> ( ) {
    list of statements
}
```

- Details
  - public – this method is part of the public specification of the class (methods can also be private; we'll see examples eventually)
  - <result type> – the type of the value returned by this query
  - <identifier> – the (hopefully meaningful) name of this method

    This is the name of the query that the method implements
  - *list of statements* – the ***body*** of the method

    These make up the algorithm that the method executes when it is called

Return Statement

- Second example of a statement

  return *expression* ;
- Meaning
  - Evaluate the expression to get a value

    In getName, the expression is just the name of the instance variable name

    For a variable, evaluation means get its current value
  - Then, finish execution of this method, replying with the value of the expression
- A value-returning method must execute a return statement to finish execution and specify the returned value

Exercise – Another Query

- **Complete the query in class HuskyCard**

```
/** Return the current balance in this HuskyCard
  *  @return the current balance in pennies. */
public int getBalance() {



}
```

Implementing Methods for Simple Commands

- **Example in class HuskyCard**

```
/** Set this HuskyCard's name to newName */
public void setName(String newName) {
    name = newName;
}
```

- **When this method is executed, it changes the name instance variable; it does not return a value**
  - **Executed only for its effect**
- **Try it out!**

More About Command Methods

- **Form**

```
/** Comment specifying the method */
public  void  <identifier> ( parameters ) {
    list of statements
}
```

- **Details**
  - **public, <identifier>, and *list of statements* – same as for queries**
  - **void – Indicates that this is a command that doesn't return a value (as opposed to the result type of a query)**

    (We can also have commands that return a result – in that case replace void with the type of the result)
  - **parameters – information supplied with command message**

    (We can also have commands with no parameters if that makes sense)

Exercise – Another Simple Command

- **Complete the command in class HuskyCard**

```
/** Set this HuskyCard's balance to newBalance */
public void setBalance(int newBalance) {



}
```

Deposit – Another Command

- In class HuskyCard

```
/** Deposit given amount in this HuskyCard */
public void deposit(double amount) {
    balance = balance + amount;
}
```

- Meaning is clear since expression in assignment statement is evaluated before balance is changed

Transfer – Objects as Parameters

- From class HuskyCard

```
/** Transfer the given amount from otherCard to this HuskyCard */
public void transfer(int amount, HuskyCard otherCard) {
    balance = balance + amount;
    otherAccount.withdraw(amount);
}
```

toString()

- Most classes should have a toString() function that returns a string with whatever state information about the object seems helpful
  - Useful in debugging, other contexts

```
/** Return a string representation of this HuskyCard
 *  @return a string identifying this as a HuskyCard with name, id, balance */
public String toString() {
   return "HuskyCard[name = " + name + ", id = " + id + ", balance = " +
                 balance + "]";
```

  - + applied to strings returns a string that has copies of the original strings pasted together

Summary

- Implementation of classes
  - Instance variables – type plus name
  - Methods – statements that make up the body of each method
- Statements
  - return
  - Assignment & arithmetic expressions
- Creating objects and calling methods

- Coming attractions
  - More details about objects, method calls, and variables
  - More complex statements – conditionals and loops