

From Last Time...

- Library Circulation system
- · Class CirculationItem class with common information
- State: title, call number, and whether checked out
- $\boldsymbol{\cdot}$ Methods: retrieve title, call number; check in and out, etc.
- Class Book extended version of CirculationItem
- Additional state author
- Additional methods get author
- Class Journal extended version of CirculationItem
- Additional state list of articles
- · Additional methods get/set list of articles

12/9/2003

(c) 2001-3, University of Washington

V-3

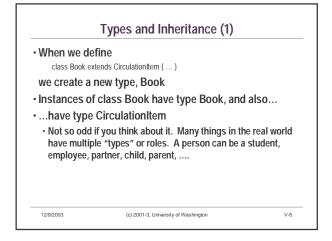
Types (Review) • Everything in Java has a type • A combination of state and operations • Primitive Types: int, double, char, boolean, ... • Simple, atomic state • Operations built in to Java language: +, -, *, /, %, &&, ||, !, ... • All other types – references to objects (class instances): Rectangle, Color, Pixel, CirculationItem, Book, ... • State is collection of instance variables • Operations are methods

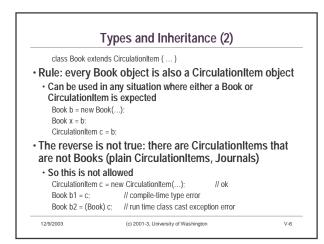
Each class definition specifies a new type with t name

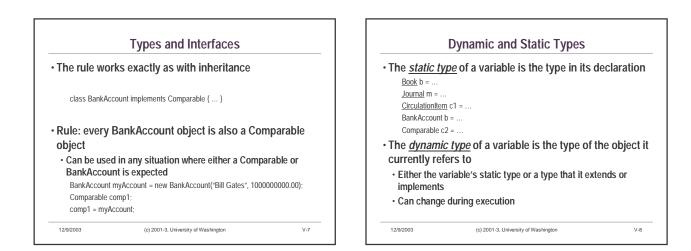
(c) 2001-3, University of Washington

12/9/2003

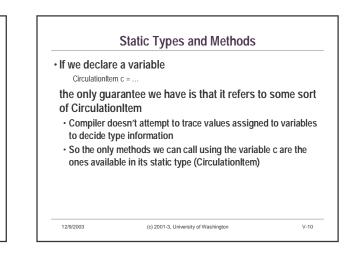
V-4



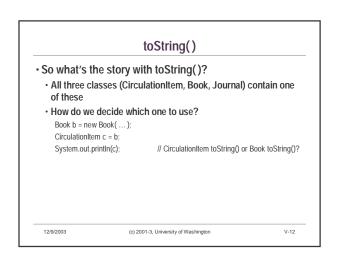


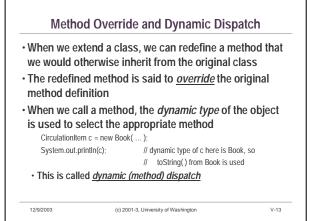


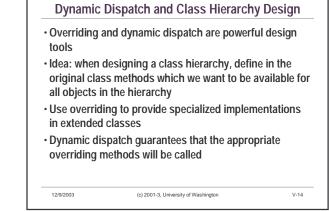
	Dynamic Types		
What are the following code	dynamic types of the variables in th de?	ie	
Book b = new B	Book("Short Story", "A. U. Thor", "P34.56");		
CirculationItem	c = new CirculationItem("Rather Bland", "A1");		
CirculationItem	d = new Journal("Long 'n Boring", "Q45.367");		
c = b;			
12/9/2003	(c) 2001-3, University of Washington	V-9	



Exam	ple	
The following produces a con	npile-time type error	
Book b = new Book("Exciting", "Great A	uthor", "H396.47");	
CirculationItem c = b;	// fine	
System.out.println(c.getAuthor());	// no – static type of c doesn // a getAuthor() method	rt include
 But if we're sure it will really be 	be a Book at runtime,	we
can use a cast		
Book temp = (Book)c;	// ok	
System.out.println(temp.getAuthor());	// fine – temp is a Book	
or		
System.out.println(((Book) c).getAuthor(()); // also ok	
12/9/2003 (c) 2001-3. University	of Washington	V-11







C	ass	Obje	ect

- The Java class structure has a root class: Object
- All Java classes implicitly extend Object if they don't explicitly extend some other class (which itself extends Object directly or indirectly) class CirculationItem {...}

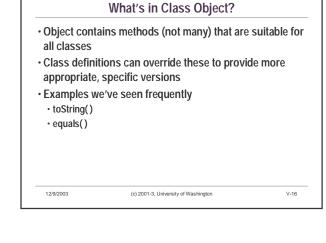
means exactly the same thing as class CirculationItem extends Object { ... }

 Classes like ArrayList have parameters and results of type Object, so will handle any non-primitive type public void add(Object obj) { ... } public Object get(int position) { ... }

12/9/2003

(c) 2001-3, University of Washington

V-15



Overloading

 In a class, it is possible to define more than one method with the same name class Thing { /** do something interesting with a Rectangle */ public void dolt(Rectangle r) { ... }

/** do something interesting with an int */ public void dolt(int n) { ... }

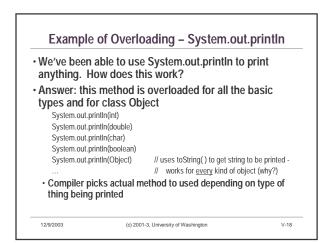
12/9/2003

- This is called method <u>overloading</u>
 Not the same thing as method overriding
 - (overriding is substituting a new method for one that would otherwise be inherited when we extend a class)

V-17

Compiler picks right method to use by comparing call argument types with parameters of available methods

(c) 2001-3, University of Washington



That's (almost) It!				
Key ideas				
 Class definit 	tion by extension ("is-a")			
 Inheritance 				
 Static and d 	ynamic types			
 Method over 	rriding			
 Dynamic dis 	patch			
 Method over 	0			
 Class Objec 	t			
Still to do				
Abstract cla	sses			
 Interfaces 				