Java Application Design
Fall 2019

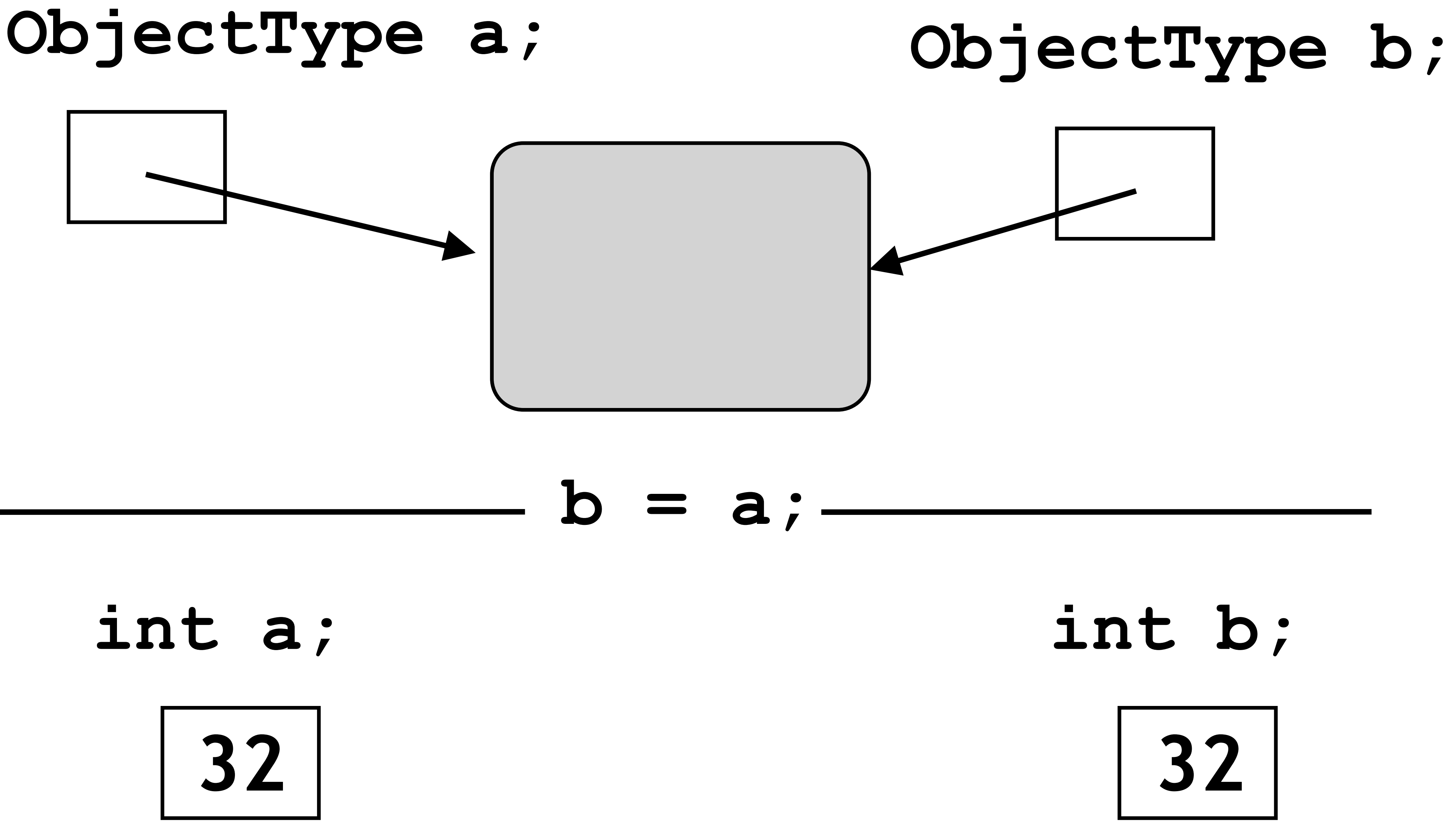
Class

Weng Kai

Primitive Data Types

Type	Storage	Min Value	Max Value
`byte`	8 bits	–128	127
`short`	16 bits	–32,768	32,767
`int`	32 bits	–2,147,483,648	2,147,483,647
`long`	64 bits	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32 bits	Approximately –3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
`double`	64 bits	Approximately –1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Primitive types vs. object types



Assignment

- Assignment "from one object to another“ is to copy a handle from one place to another
  - Case: Assignment.java

Passing by Value(I)

```
void f(int n) {
   n = 9;
}
int k;
k = 10;
f(k);
```

Passing by Value(II)

```
class Number {
   public int i;
}
void f(Number n) {
   n.i = 9;
}
Number k = new Number();
k.i = 10;
f(k);
```

Passing by Value(III)

```
void f(int n) {
n = 9;
}
int k;
k = 10;
f(k);
```

```
class Number {
   public int i;
}
void f(Number n) {
   n.i = 9;
}
Number   k   =   new
Number();
k.i = 10;
f(k);
```

Relation

- Relation operators == != can deal with any objects, but…
  - Case: Equivalence.java
  - Case: EqualsMethod.java
  - Case: EqualsMethod2.java

this

- The keyword **this** produces the reference to the object the method has been called for.
  - Case: Leaf.java
- **this** can make an explicit call to another constructor.
  - Case: Flower.java

Cleanup: finalize()

- When the garbage collector is ready to release the storage used for your object, it will first call its **finalize()**.
- But finalize() is totally different from destructor of C++

1. Garbage collection is not destruction.
2. Your objects might not get garbage-collected.
3. Garbage collection is only about memory.

Member initialization

Java goes out of its way to guarantee that any variable is properly initialized before it is used.

- Since any method can initialize or use that data, it might not be practical to force the user to initialize it to its appropriate value before the data is used. So each primitive data member of a class is guaranteed to get an initial value -- 0.

Specifying initialization

```
class Measurement {
    boolean b=true;
     char c='x';
     int i=47;
 };

class Measurement {
    Depth o = new Depth();
};
```

```
class CInit {
    int i = f(); // ...
}

class CInit {
    int i = f(); int k = g(i); // ...
};

class CInitWrong {
     int j = g(i); int i = f(); // ...
};
```

Order of initialization

- Within a class, the order of initialization is determined by the order that the variables are defined within the class.
-
- Case: OrderOfInitialization.java

Static data initialization

- Static members are members of class
- Class Class and its instances
- Static member is to be initialized in the loading of the class
- Case: StaticInitialization.java

Order of initialization

- The first time an object of type **Dog** is created, *or* the first time a **static** method or **static** field of class **Dog** is accessed, the Java interpreter must locate **Dog.class**,
- As **Dog.class** is loaded, all of its **static** initializers are run.
- When you create a **new Dog( )**, the construction process for a **Dog** object first allocates enough storage for a **Dog** object on the heap.
- This storage is wiped to zero, automatically setting all the primitives in that **Dog** object to their default values.
- Any initializations that occur at the point of field definition are executed.
- Constructors are executed.

Explicit static initialization

- Java allows you to group other **static** initializations inside a special "**static** construction clause" (sometimes called a *static block*) in a class.

Case: ExplicitStatic.java

- Java provides a similar syntax for initializing non-**static** variables for each object

Case: Mugs.java

Package

- The import keyword is to bring in an entire library or a member of that one

```
import java.util.*;

import java.util.Vector;
```

- Package is to provide a mechanism to manage “name spaces.”

compilation unit

- Each compilation unit must have a name ending in .java, and inside the compilation unit there can be a public class that must have the same name as the file. There can be only one public class in each compilation unit.
- When you compile a .java file you get an output file with exactly the same name but an extension of .class for each class in the .java file.
- A working program is a bunch of .class files

Define a package

- A library is also a bunch of these class files.

```
  package mypackage;

  public class MyClass;
```

- if someone wants to use MyClass, they must use the import keyword to make the name or names in mypackage available.

```
import mypackage.*;

MyClass m = new MyClass();
mypackage.MyClass m = new
mypackage.MyClass();
```

CLASSPATH

- Place all the .class files for a particular package into a single directory
- CLASSPATH contains one or more directories that are used as roots for a search for .class files.

Java access specifiers

- placed in front of each definition for each member in your class, whether it’s a field or a method.
  - “friendly” (default package)
  - `public`
  - `private`
  - `protected`: “sort of friendly”

Class Access

- There can be only one public class per compilation unit (file).
- The name of the public class must exactly match the name of the file.
- It is possible, though not typical, to have a compilation unit with no public class at all. In this case, you can name the file whatever you like.
- CASE: Lunch.java

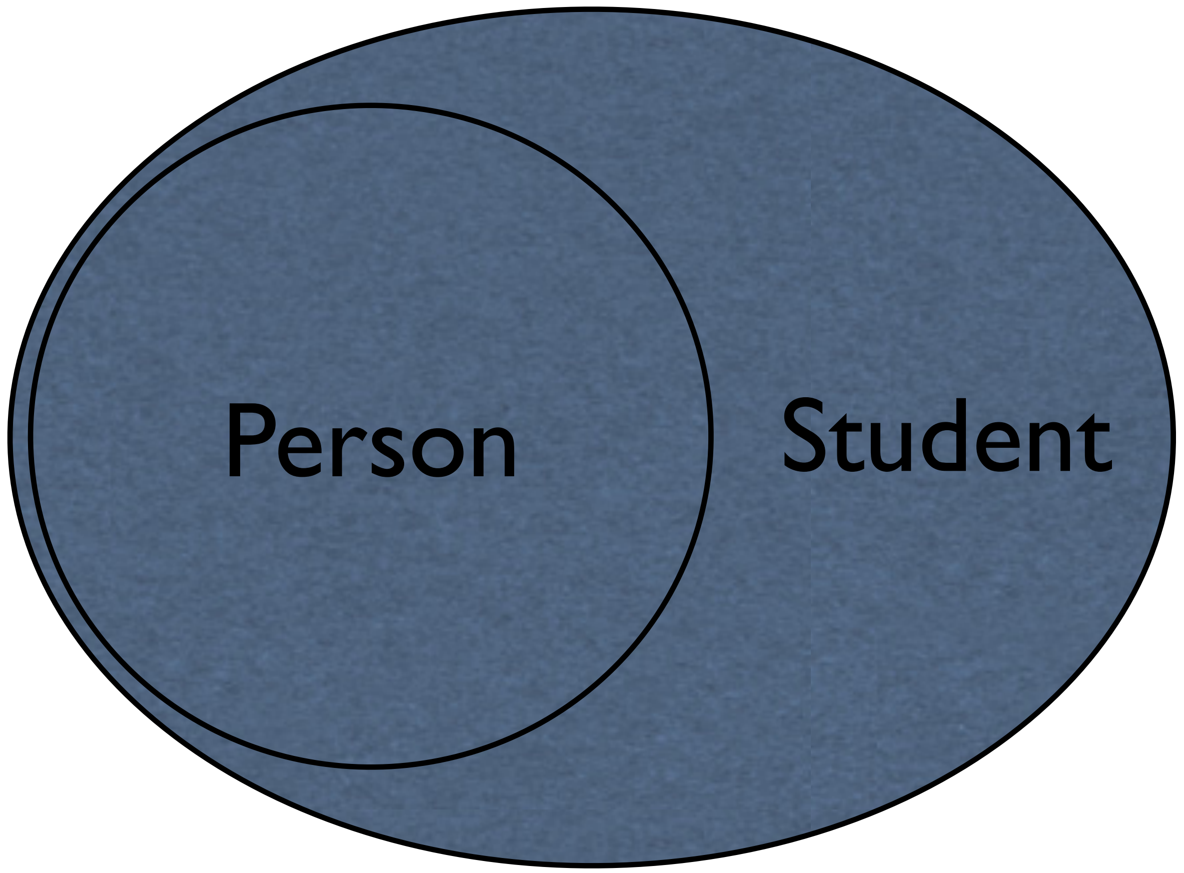
Inheritance

- An important component of the OO design methodology
- Allows sharing of
  - fields
  - methods

  between classes
- Key technology in Java

Inheritance

- The ability to define the behavior or implementation of one class as a superset of another class



Inheritance syntax

```
class ThisClass extends
SuperClass {
    //    class body

}
```

- Case Study: Detergent.java

Inheritance & Ctors

- Think of inherited traits as an embedded object
- A part of an object of the subclass is exact the object of the superclass
- So it must be initialized b4 the initialization of the sub one

Inheritance & Ctors

- Base class is always constructed first
- If no explicit arguments are passed to base class, default constructor will be called
- Case: Cartoon.java

Inheritance & Ctors

- For super's ctor w/ arguments, use keyword super to call the ctor and pass value to it.
- Case: Chess.java

Name hide?

- What if you define a method w/ same name as one in superclass but different parameter list?
- Case: Hide.java

What is not inherited?

- Constructors
- Private data is hidden, but still present

- Case: Son.java

Initialization & class load

- Now we have:
  - Static init., member init. and ctor for base
  - Static init., member init. and ctor for derived

  What is the sequence to do all these init?

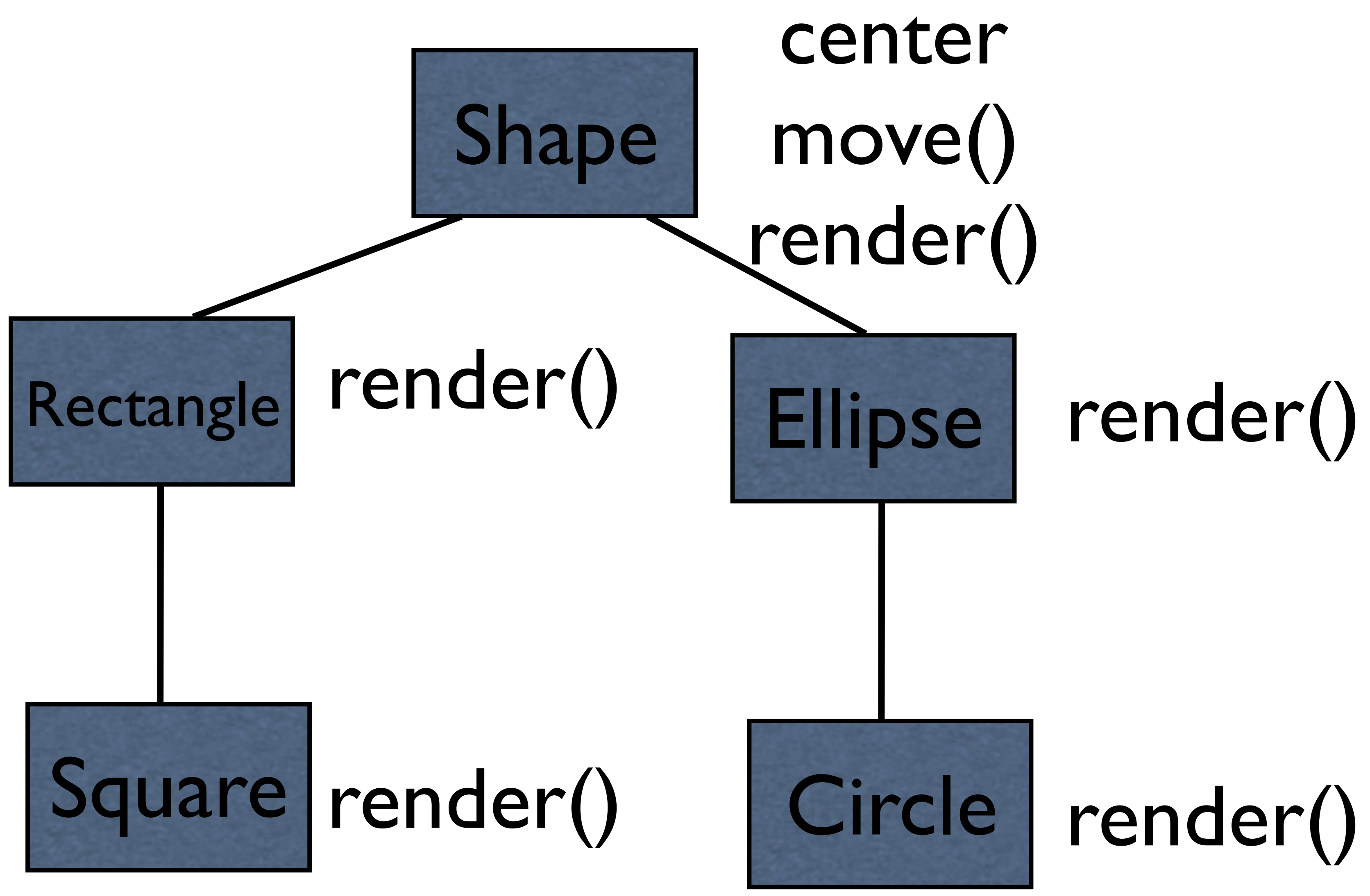
- Case: Beetle.java

Upcast

- is taking an object handle and treating it as the handle of its base type.
- The relationship between sub and base classes is :The new class is a type of the existing class.

- Case: Wind.java

Conceptual model



Method call binding

- is connecting a method call to a method body: Which function is to be called?
  - Static binding: call the function as the code
  - Dynamic binding: call the function of the object
- Case: Shapes.java

Override

- Methods in super and sub classes have same name AND same parameters.
- NOT for fields, ctors. and private methods.

- Case: Override.java

final

- final : This cannot be changed
  - final for data
  - final for methods
  - final for a class

Final fields

- Case Study: FinalData.java
- Case Study: BlankFinal.java
- Case Study: FinalArguments.java

Final methods

- two reasons for final methods:
  - To put a "lock" on the method to prevent any inheriting class from changing its meaning.
  - The compiler is to turn any calls to that method into inline calls.
- private is final.

Final classes

- you don’t want to inherit from this class or allow anyone else to do so.
- Discuss:
  - Final fields in a final class?
  - Final methods in a final class?

Abstract classes and methods

- Some class is to create a common interface for all the classes derived from it.
- An abstract method is incomplete. It has only a declaration and no method body.
- A class containing abstract methods is called an abstract class.
- Case : Music4.java

Interface

totally abstract class

- All methods in interface are public.
- All data members in interface are public static final.

Interface

- To define an interface:

```
public interface InterfaceName extends BaseInterfaces
```

- To inherit from an interface:

```
class ClassName implements interfaces
```

- Case: Music5.java

Use of interface

- An interface can be used as a class

  Case: Store.java

- An interface can inherit from more than one interfaces, but not class.

  Case: HorrorShow.java

- A class can implement more than one interfaces.

  Case: Adventure.java