Chapter 3.

Primitives and References

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

Variable Declaration

- Java cares about type (e.g., it is illegal to put a floating point number into an integer variable)
- All variables must be declared before use
- A variable is declared by specifying its type and name

type name

```
int x;
```

- The type of a variable specifies the kind of data that can be stored in the variable
- The name of a variable is used for referring to the variable

Naming a Variable

- Simple rules in naming variables
  - Must start with a letter, an underscore (_), or a dollar sign ($), but not with a number
  - The rest of the characters may be letters, underscores, dollar signs and numbers
  - Cannot be a reserved word in Java
- Reserved words in Java

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

Standard Naming Conventions

- Long meaningful names make the source code more readable
- Follow standard naming conventions
  - A class name begins with a capital letter (e.g., Dog)
  - A variable or method name begins with a lowercase letter (e.g., size, makeNoise())
  - The name of a symbolic constant consists of only capital letters (e.g., PI, GRAVITY)
  - Names consisting of multiple words are joined together with each subsequent word begins with a capital letter (e.g., ShoppingCart, addItemToCart())

Data Types

- The data type of a variable determines
  - The memory space allocated to the variable
  - How the computer interpret the data stored in the variable
- In Java, data types can be classified into
  - Primitives – fundamental values including integers, booleans and floating point numbers
  - Object references – references to objects

Primitive Types

- Java supports 8 primitive types

	Type	Bit Depth	Value Range
	boolean	JVM-specific	true or false
	char	16 bits	0 to 65535
integer	byte	8 bits	–128 to 127
integer	short	16 bits	–32768 to 32767
integer	int	32 bits	–2147483648 to 2147483647
integer	long	64 bits	–huge to huge
floating point	float	32 bits	varies
floating point	double	64 bits	varies

Declarations and Assignments

- Examples

```
boolean isFun = true;
char c = 'f';
byte b = 89;
int x = 234;
long big = 3456789;
float f = 32.5f;
double d = 3456.98;
int y = x;
int z = 3 * y;
```

Java treats all numbers with a floating point as double. The 'f' here is used to specify that this number is a float

- Values that can be assigned to a primitive variable
  - Literal value of a compatible type
  - Value of another variable of a compatible type
  - Value (of a compatible type) returned by an expression

Type Conversion

- The compiler does not allow assigning a value of a data type with a wider range to a variable declared with a data type with a narrower range (as this might result in information loss)
- Hence the following assignment statements are illegal

```
int x = 24;
byte b = x; // type mismatch
float f = 32.5; // type mismatch
```

- Type casting tells the compiler it is an intended type conversion and prevents the compiler from reporting an error

```
int x = 24;
byte b = (byte) x;
float f = (float) 32.5;
```

type casting

Type Conversion

- Care must be taken when carrying out type casting as information may be lost
- Example

```
public class TypeCastingExample {
        public static void main(String[] args) {
                int x = 40000;
                byte b = (byte) x; // type casting
                System.out.println("b = " + b);
        }
}
```

- Sample output

```
b = 64
```

$x = 40000 = (1001110001000000)_2$. When x is cast into a byte, only the right most 8 bits are kept, i.e., $(01000000)_2 = 64$. Hence, b is assigned a value of 64.

Object References

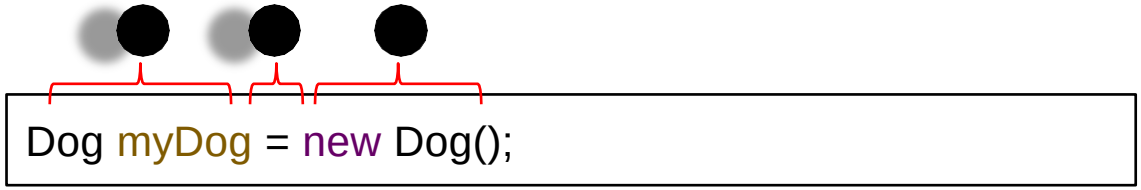
- There are only object reference variables (or simply reference variables), but not object variables
- A primitive variable holds bits that represent the actual value of the variable
- A reference variable
  - Holds bits that represent a way to access a specific object (i.e., an object reference)
  - Does not hold the object itself (objects live on the heap!)
- An object reference is something similar to a pointer or an address, and is used by JVM to get to the object (unlike pointers or addresses in C++, no arithmetic is allowed on object references)

Object References

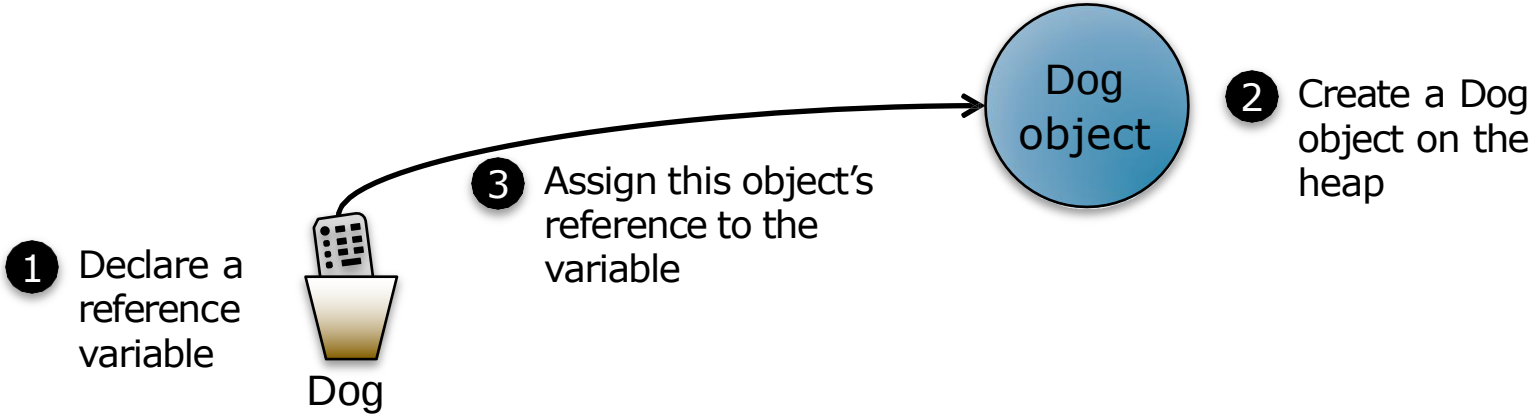
- A reference variable is like a remote control
- Using the dot operator on a reference variable is like pressing a button on the remote control to access a method or instance variable of an object
- A reference variable has a value of null when it is not referencing any object

```
Dog myDog = new Dog();
myDog.makeNoise();
```

Declaration, Creation and Assignment



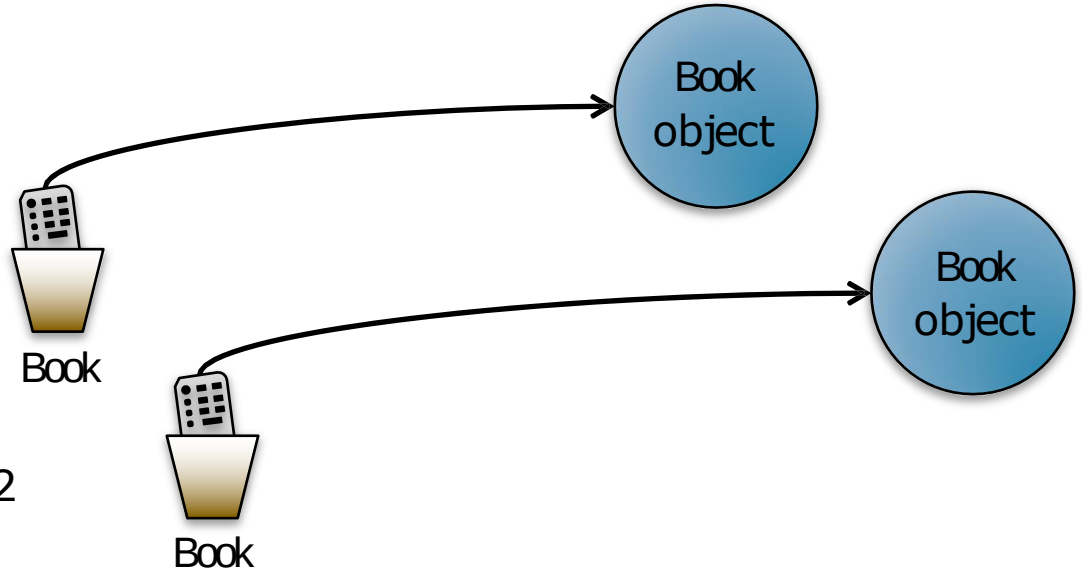
- The 3 steps in object declaration, creation and assignment
  1. Declare a reference variable
  2. Create an object on the heap
  3. Assign this object's reference to the variable



Life and Death on the Heap

- Example

```
Book b = new Book();
Book c = new Book();
```

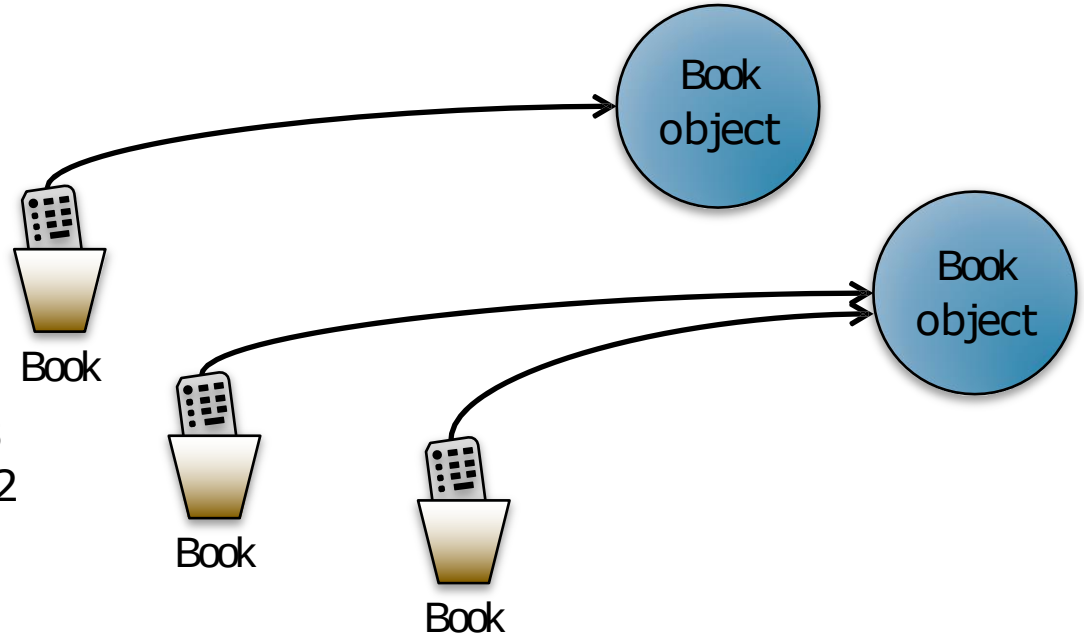


active references: 2
reachable objects: 2

Life and Death on the Heap

— Example

```
Book b = new Book();
Book c = new Book();
Book d = c;
```

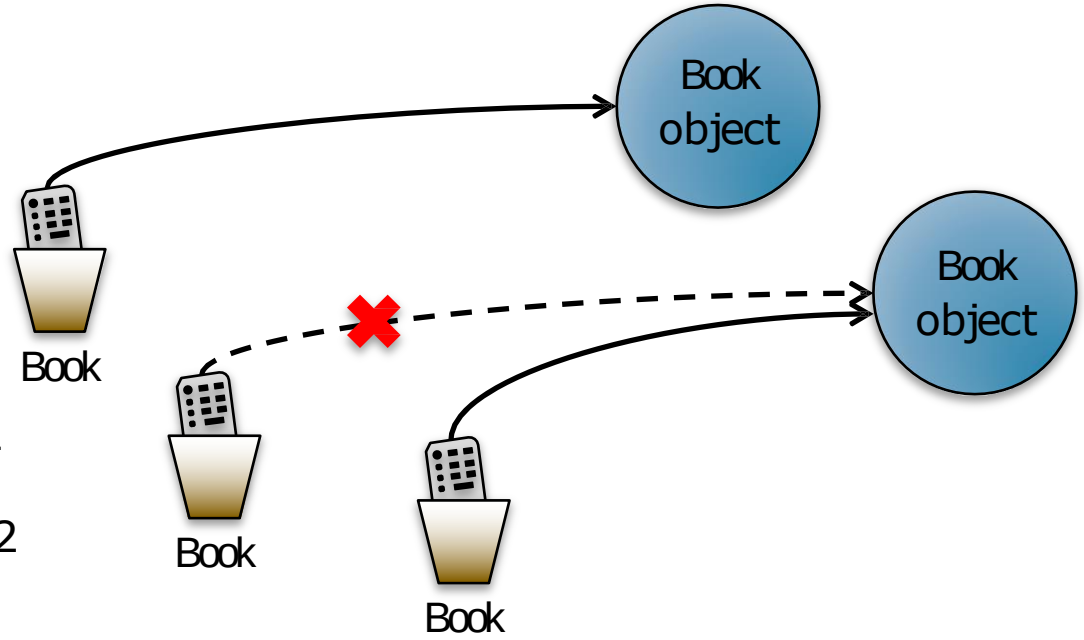


active references: 3
reachable objects: 2

Life and Death on the Heap

— Example

```
Book b = new Book();
Book c = new Book();
Book d = c;
c = null;
```



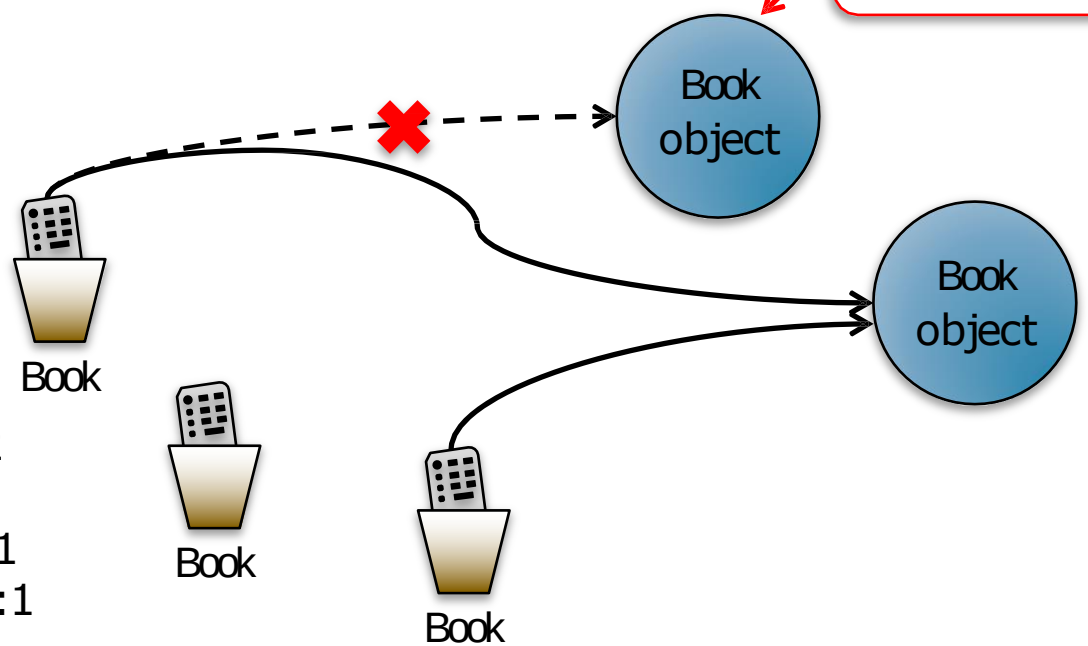
active references: 2
null references: 1
reachable objects: 2

Life and Death on the Heap

— Example

```
Book b = new Book();
Book c = new Book();
Book d = c;
c = null;
b = d;
```

This object becomes eligible for garbage collection



active references: 2
null references: 1
reachable objects: 1
abandoned objects:1

Arrays

- An array is a collection of data of the same type
- Like in C++
  - The size of an array must be determined at the time of creation, and cannot be changed afterwards
  - The array index is zero-based
  - Each array element can be accessed using its index with the subscript operator []
- In Java, arrays are objects which have instance variables and methods (e.g., each array object has an instance variable named length that stores its size)

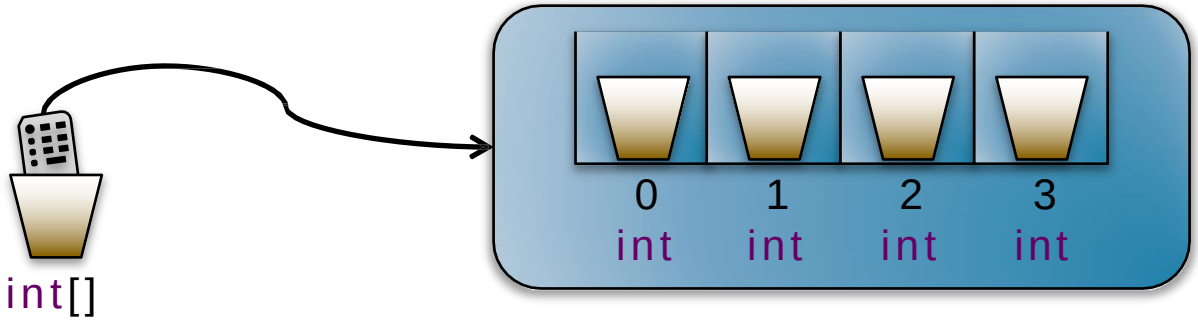
Arrays

- Every element in an array is just a variable, i.e., a variable of 1 of the 8 primitive types or a reference variable
- Anything that can be put into a variable of that type can be assigned to an array element of that type
- Examples
  - In an `int` array (`int[]`), every element is an `int` variable that can hold an `int` value (an array object can have elements which are primitives, but the array itself is never a primitive!)
  - In a Dog array (`Dog[]`), every element is a reference variable that can hold a reference to a Dog object (a reference variable holds a reference to an object, but not the object itself!)

An Array of Primitives

- Example

```
int[] nums;
nums = new int[4];
nums[0] = 6;
nums[1]  = 19;
nums[2]  = 44;
nums[3]  = 42;
```

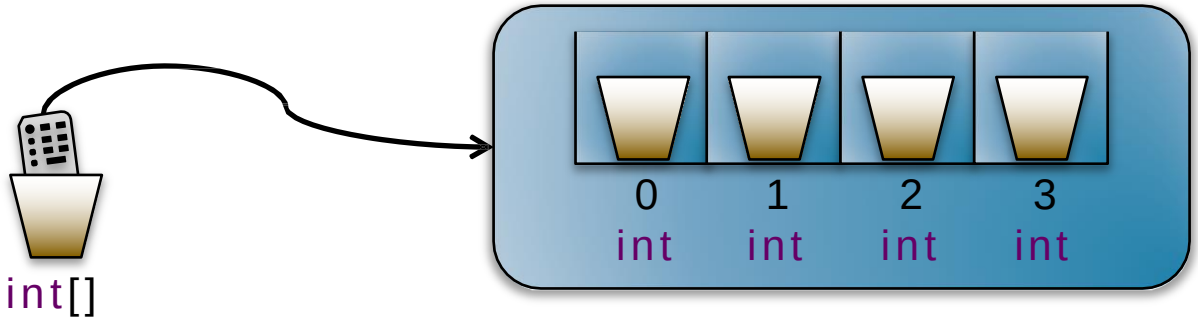


An Array of Primitives

- Example

```
int[] nums = {6, 19, 44, 42};
```

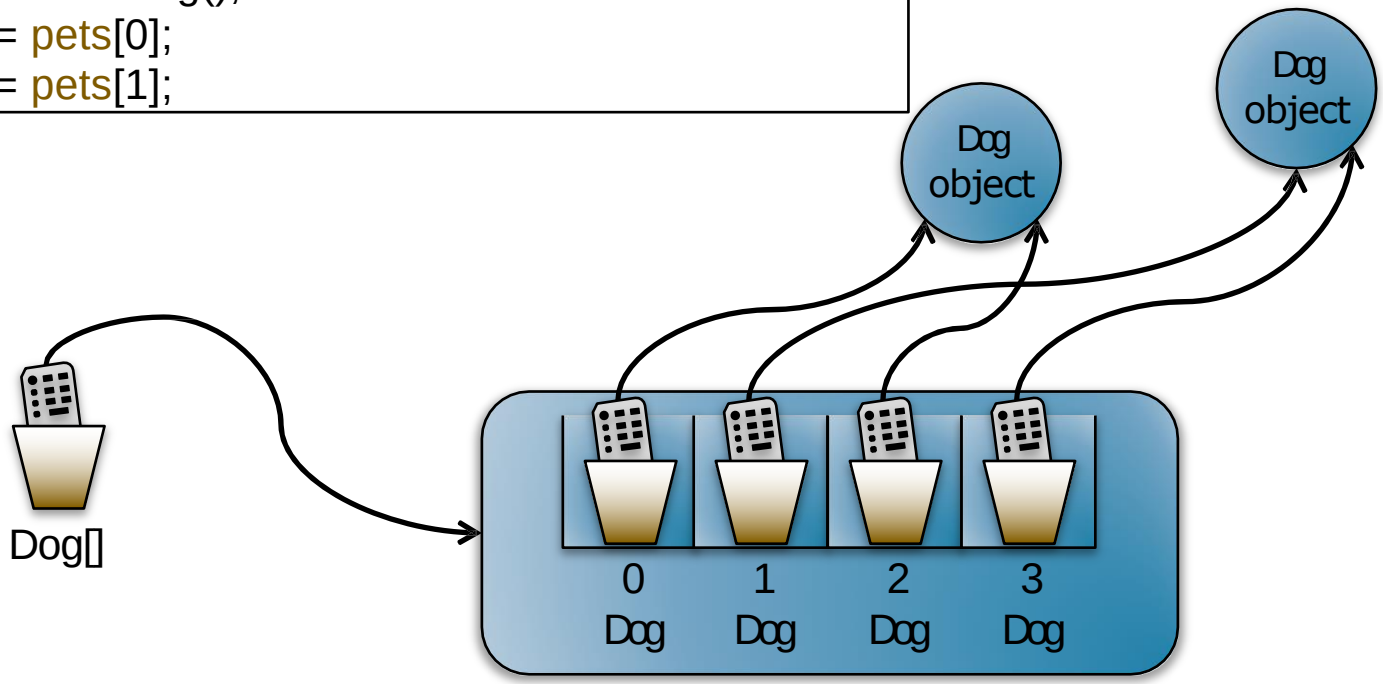
A shortcut syntax for creating and initializing an array. The size of the array is determined by the number of initialization values



An Array of Objects

- Example

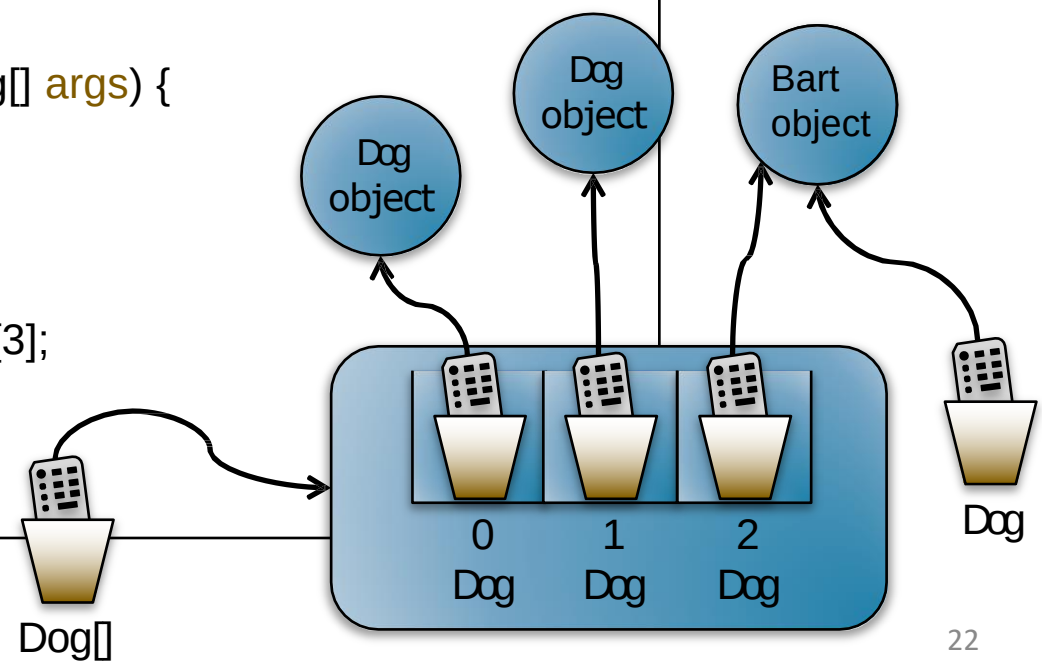
```
Dog[] pets;
pets = new Dog[4];
pets[0] = new Dog();
pets[1] = new Dog();
pets[2] = pets[0];
pets[3] = pets[1];
```



Example: An Array of Dogs

— Example

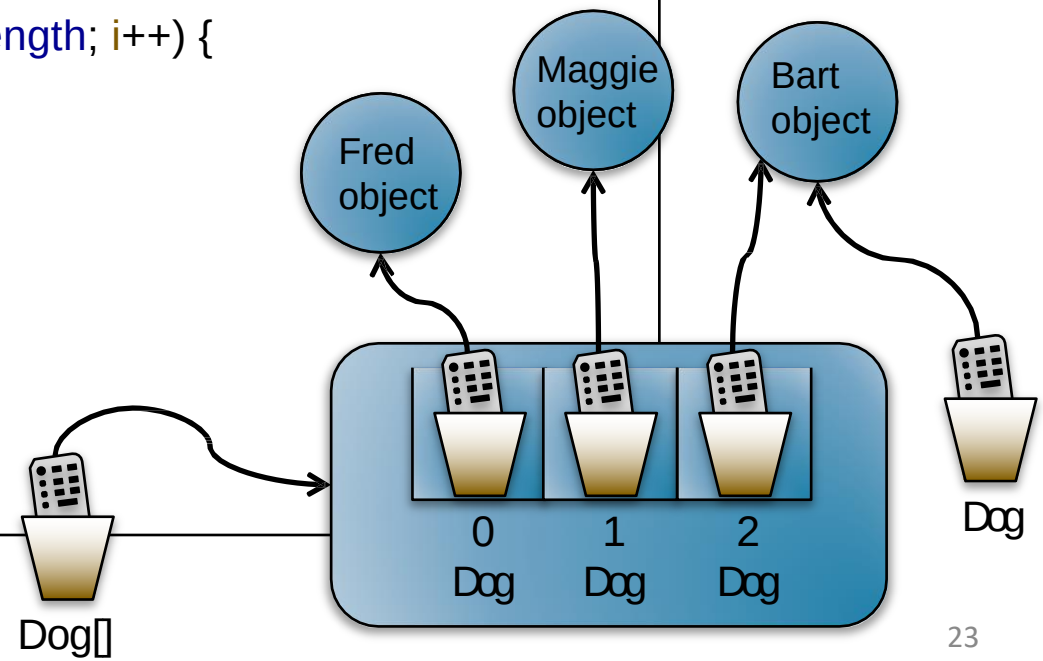
```
class Dog {
   String name;
   public void makeNoise() {
      System.out.println(name + " says Woof!");
   }
}

   public static void main(String[] args) {
      Dog dog1 = new Dog();
      dog1.makeNoise();
      dog1.name = "Bart";

      Dog[] myDogs = new Dog[3];
      myDogs[0] = new Dog();
      myDogs[1] = new Dog();
      myDogs[2] = dog1;
```



Example: An Array of Dogs

— Example

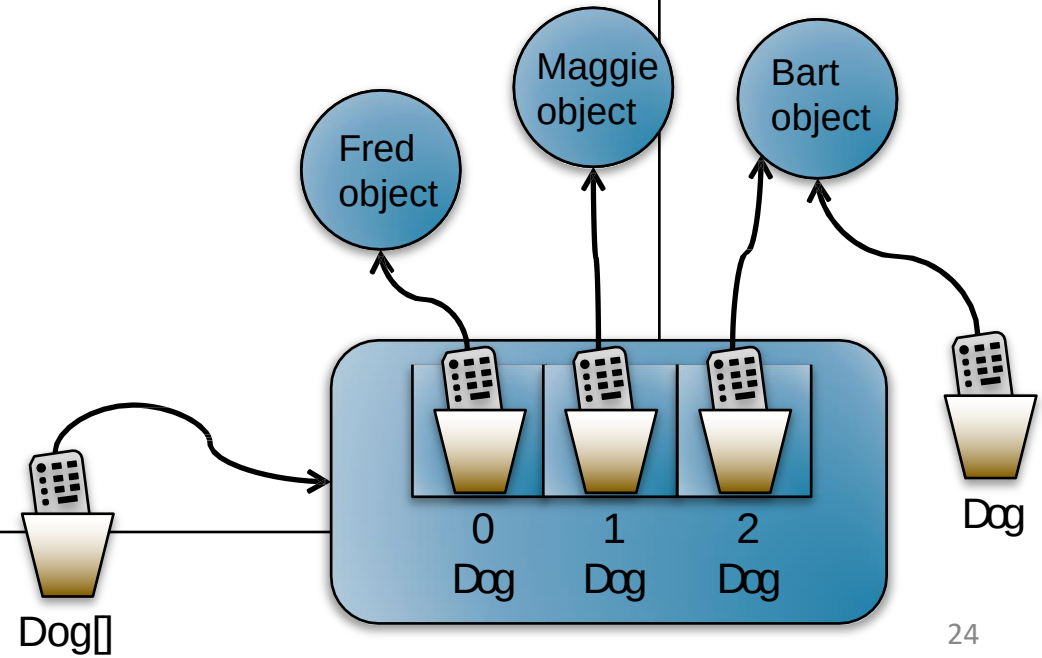
```
        myDogs[0].name = "Fred";
        myDogs[1].name = "Maggie";

        System.out.print("last dog's name is ");
        System.out.println(myDogs[2].name);

        for (int i = 0; i < myDogs.length; i++) {
            myDogs[i].makeNoise();
        }
    }
}
```



Example: An Array of Dogs

- Sample output

```
null says Woof!
last dog's name is Bart
Fred says Woof!
Maggie says Woof!
Bart says Woof!
```



Multidimensional Arrays

- It is possible to create an array of array (aka multi-dimensional array)
- Example

```
public class MultiDimArray {
   public static void main(String[] args) {
      String[][] names = {
         {"Mr. ", "Mrs. ", "Ms. "},
         {"Smith", "Jones"}
      };
      System.out.println(names[0][0] + names[1][0]);
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

- Sample output

```
Mr. Smith
Ms. Jones
```

Multidimensional Arrays

- Example

```
public class MultiDimArray2 {
   public static void main(String[] args) {
      String[][] names = new String[2][3];
      names[0][0] = "Mr. ";
      names[0][1] = "Mrs. ";
      names[0][2] = "Ms. ";
      names[1][0] = "Smith";
      names[1][1]  = "Jones";
      System.out.println(names[0][0] + names[1][0]);
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

Is names in MultiDimArray2 identical to names in MultiDimArray?

Multidimensional Arrays

— Example

```
public class MultiDimArray2 {
   public static void main(String[] args) {
      String[][] names = new String[2][3];
      names[0][0] = "Mr. ";
      names[0][1] = "Mrs. ";
      names[0][2] = "Ms. ";
      names[1][0] = "Smith";
      names[1][1]  = "Jones";
      System.out.println(names[0][0] + names[1][0]);
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

Is names in MultiDimArray2 identical to names in MultiDimArray?

No! In MultiDimArray, names[0].length is 3 and names[1].length is 2. In MultiDimArray2, both names[0].length and names[1].length are 3!

Multidimensional Arrays

- Example

```
public class MultiDimArray3 {
   public static void main(String[] args) {
      String[][] names = new String[2][];
      String[] titles = {"Mr. ", "Mrs. ", "Ms. "};
      String[] surnames = {"Smith", "Jones"};
      names[0] = titles;
      names[1] = surnames;
      System.out.println(names[0][0] + names[1][0]);
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

Now in MultiDimArray3, names[0].length is 3 and names[1].length is 2

Chapter 3.

End

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong