CISC 3120
C04: Methods and Arguments

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap and issues
- Instance variables and methods
- Methods and arguments
- Eclipse and Maven Integration for Eclipse
- Search solutions online
- Assignments

Class

- Blueprint for
  - What an object knowns (fields)
    - Representing states
    - instance variables (non-static variables)
    - Class variables (static variables)
  - What an object does (methods)
    - Representing behaviors
    - instance methods (non-static methods)
    - class methods (static methods)

Example: the Dog class

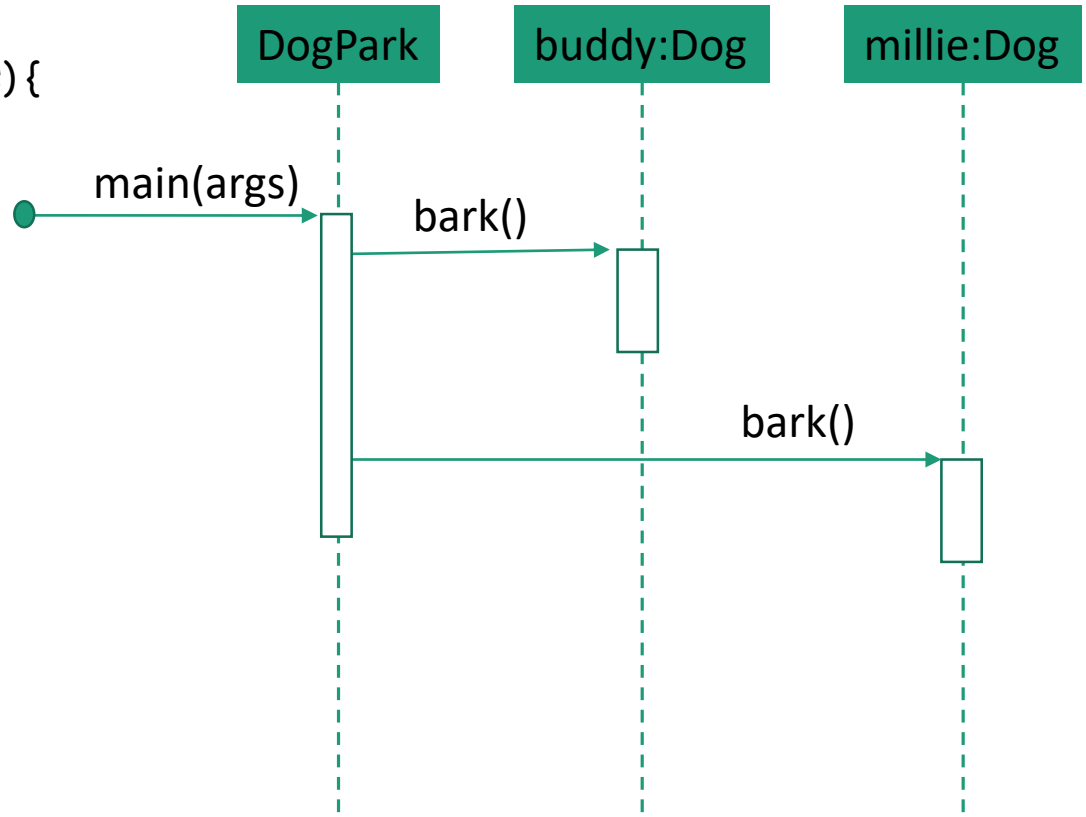
```
class Dog {
    int size;
    String name;

    void bark() {
        if (size > 60) {
            System.out.println("Woof!");
        } else if (size > 14) {
            System.out.println("Ruff!");
        } else {
            System.out.println("Yip!");
        }
    }
}
```

Object-to-Object Communication

- Representing the real world,
  - an application has many classes,
  - each can have many objects
- Object-to-object communication
  - Via method invocation (method calls, sometimes also referred to as "message passing")
    - One object invokes another object's instance method
    - One object invokes another class's class method
    - One class invokes another object's instance method
    - One class invokes another class's class method

Example: the DogPark class

```
class DogPark {
    public static void main(String[] args) {
        Dog buddy = new Dog();
        Dog millie= new Dog();
        buddy.bark();
        millie.bark();
    }
}
```



- UML: Sequence Diagram

Example: Method Invocations

- One class invokes another object's instance method
- One class invokes another class's class method

```
class SquareRoot {
   public static void main(String[] args) {
      double number = Double.parseDouble(args[0]);
      System.out.println(Math.sqrt(number));
   }
}
```

Example: Method Invocations

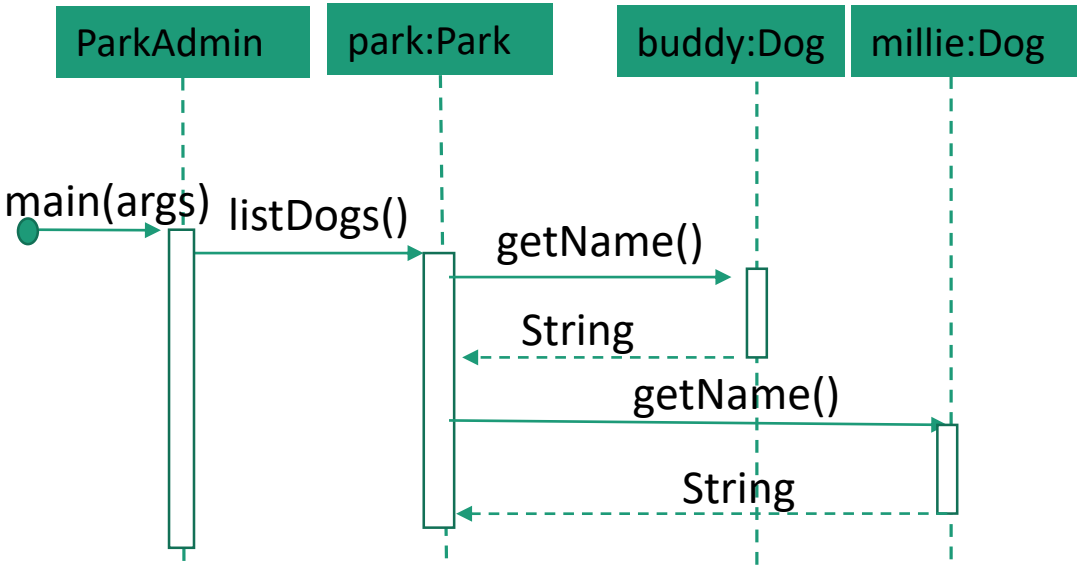
- One class invokes another object's instance method
- One class invokes another class's class method
- One object invokes another object's method

```
class DogPark {
    Dog buddy = new Dog();
    Dog millie= new Dog();
    void listDogs() {
        System.out.println(buddy.getName());  }
        System.out.println(millie.getName()); }
    }  }
```

```
class ParkAdmin  {
  public static void main(String[] args) {
      DogPark park = new DogPark();
      park.listDogs();
```

Example: Method Invocations

- One object invokes another object's method



Instance and Class Methods

- An instance method belongs to an object
- A class method belongs to class
- Question: This is illegal. Why?

```
class Cat {
    public static void main(String[] args) {
        meow();
    }
    void meow() {
        System.out.println("meow …");
    }
}
```

Instance and Class Methods

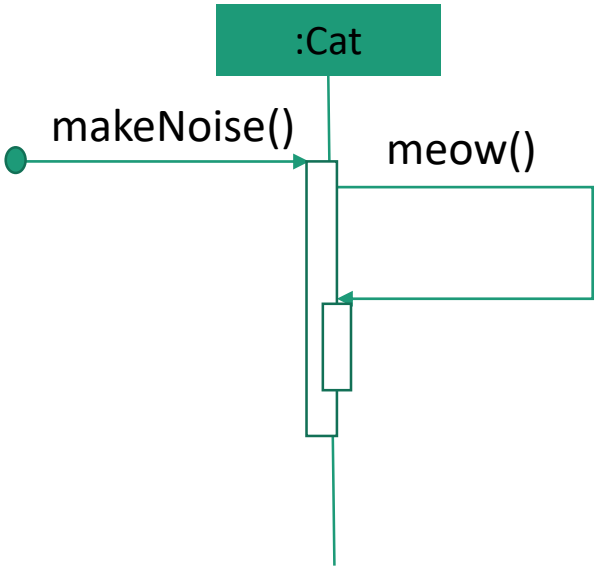
- An instance method belongs to an object
- A class method belongs to class
- Question: This is also illegal. Why?

```
class DogPark {
    Dog buddy = new Dog();
    public static void main(String[] args) {
        System.out.println(buddy.getName());
    }
}
```

Instance and Class Methods

- An instance method belongs to an object
- A class method belongs to class
- Question: However, this is OK. Why?

```
class Cat {
   void makeNoise() {
        meow();
   }
   void meow() {
        System.out.println("meow …");
    }
}
```



Questions?

- How classes and objects interact with each other?
  - Method invocation

Formal Parameters

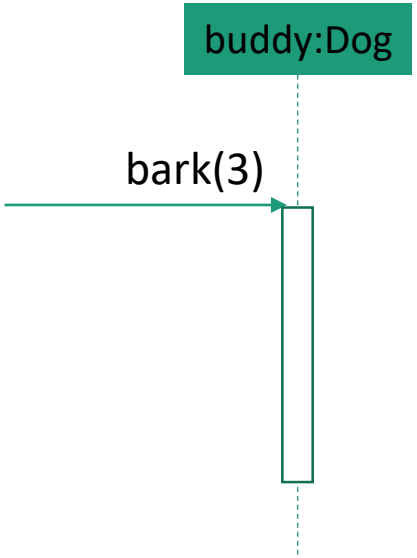
- A method can have formal parameters

```
public class Dog {
    int size;
    String name;
    void bark(int numOfBarks) {
        for (int i=0; i<numOfBarks; i++) {
            System.out.println("Ruff");
        }
    }
}
```

Passing Arguments

- Caller passes arguments to the method

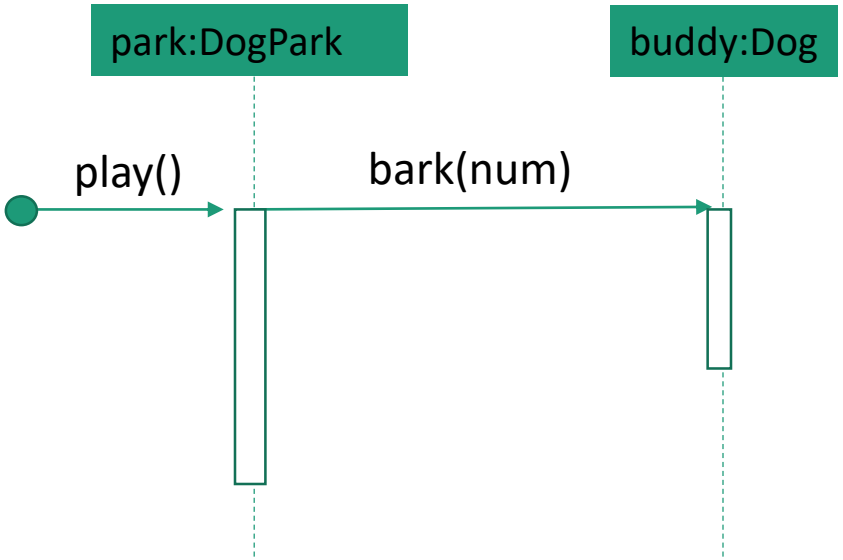
```
Dog buddy = new Dog();
buddy.bark(3);
```



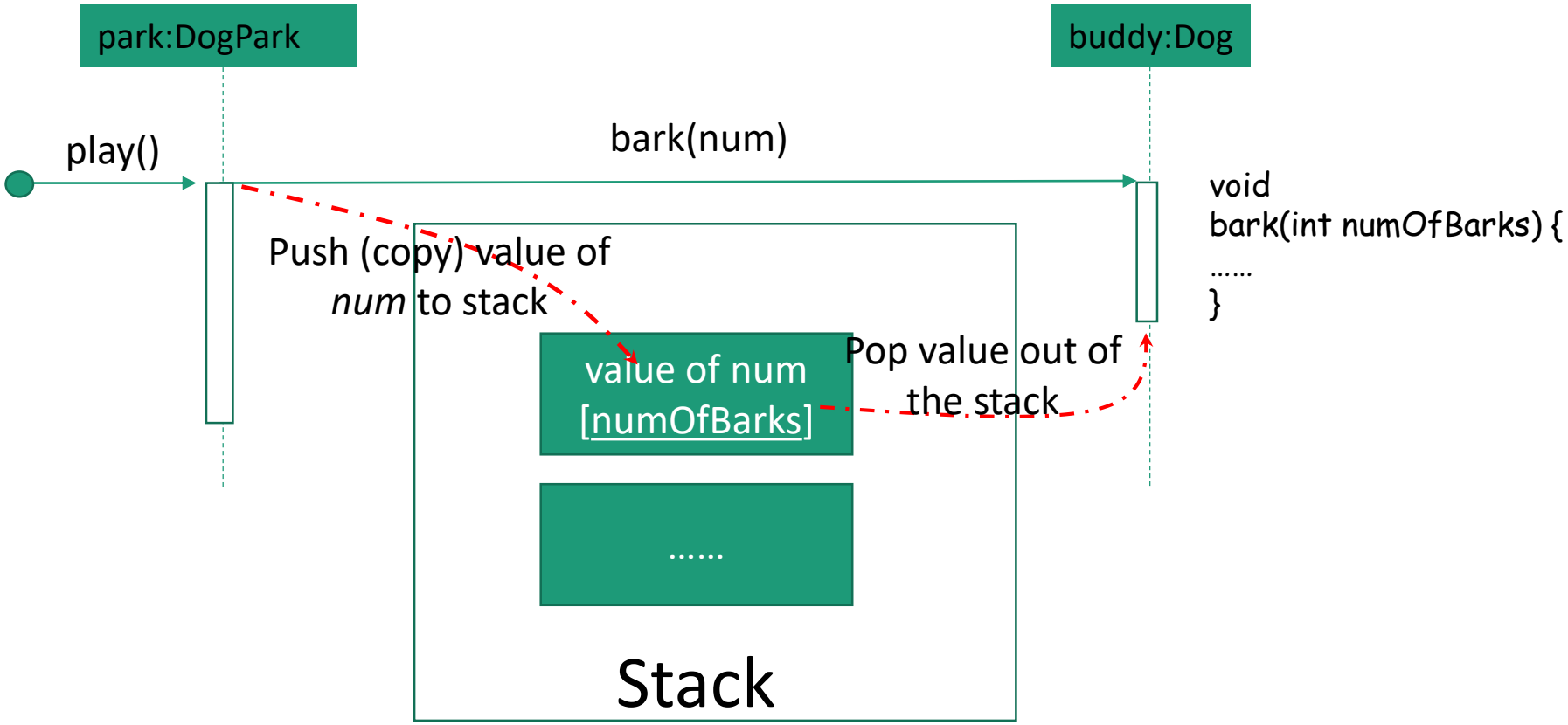
Pass-by-Value

- Java always passes an argument by its value

```
class DogPark {
      void play() {
            int num = 3;
            Dog buddy = new Dog();
            buddy.bark(num);
      }
}
```



Pass-by-Value



- Stack: also a data structure, first-in-last-out

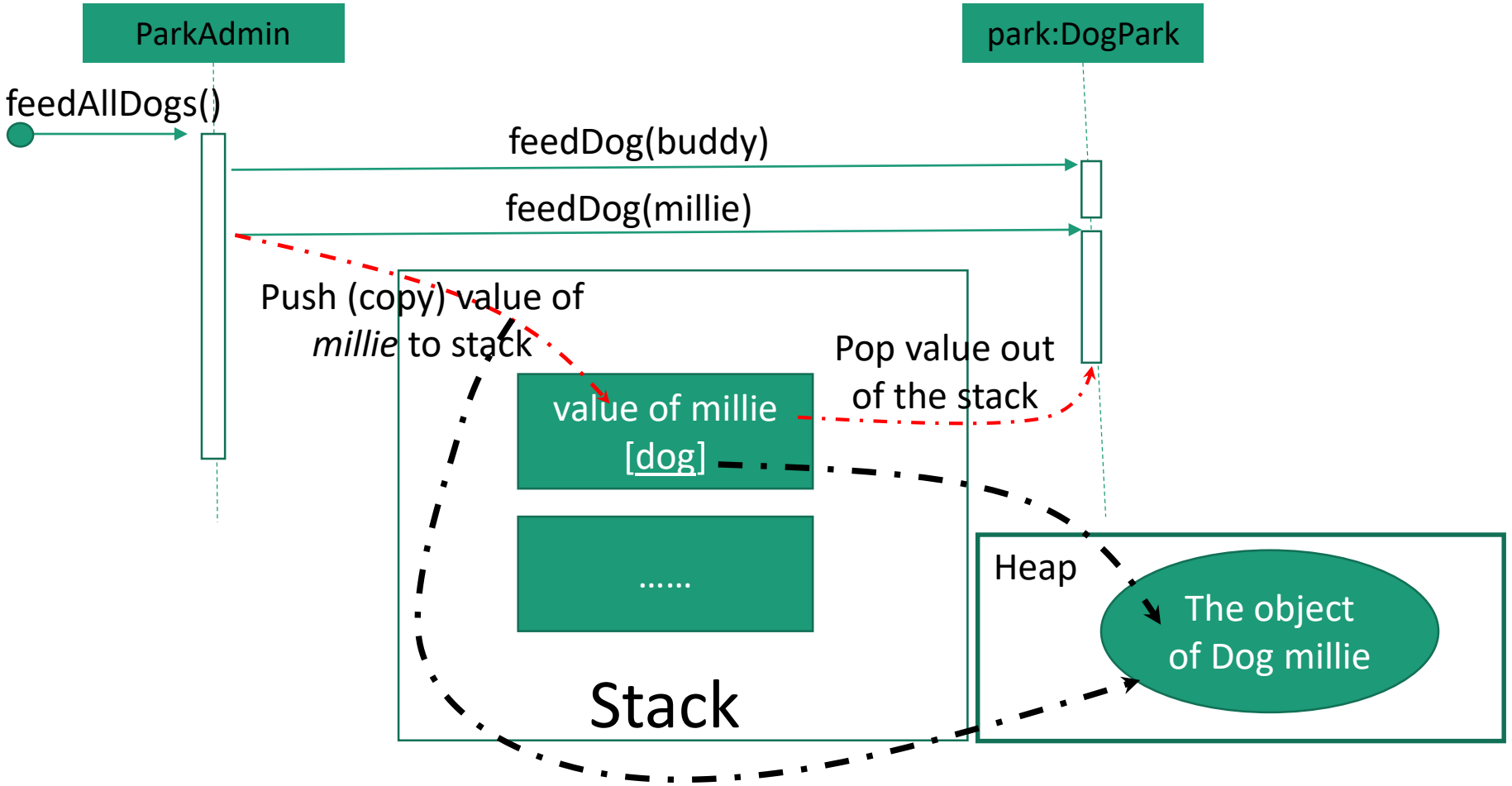
How about Reference Variables?

- The value of the reference is pushed into the Stack
- Consider adding methods to the ParkAdmin & DogPark classes

```
class DogPark {
    Dog millie = new Dog();
    ……
    void feed(Dog dog) {  …  }
}
```

```
class ParkAdmin {
    static void feedAllDogs(DogPark park) {
        Dog[] dogs = park.getAllDogs();
        if (dogs.length >= 2) {
            Dog buddy = dogs[0];
            park.feedDog(buddy);
            Dog millie = dogs[1];
            park.feedDog(millie);    }   }
}
```

Pass Value of Reference Variable



Arrays are Objects in Java

```
public class PassByValueDemo {
  void changeX(int[] x)    {   System.out.println("changeX: at beginning: x[0] = " + x[0]);
                               x[0] = 2;   System.out.println("changeX: at the end: x[0] = " + x[0]);  }
}
public class PassByValueDemoTestDrive {
   public static void main(String[] args) {
                                PassByValueDemo demo = new PassByValueDemo();
                                int[] x = {3};
                                System.out.println("main: before calling changeX: x[0] = " + x[0]);
                               demo.changeX(x);
                               System.out.println("main: after calling changeX: x[0] = " + x[0]);  }
}
```

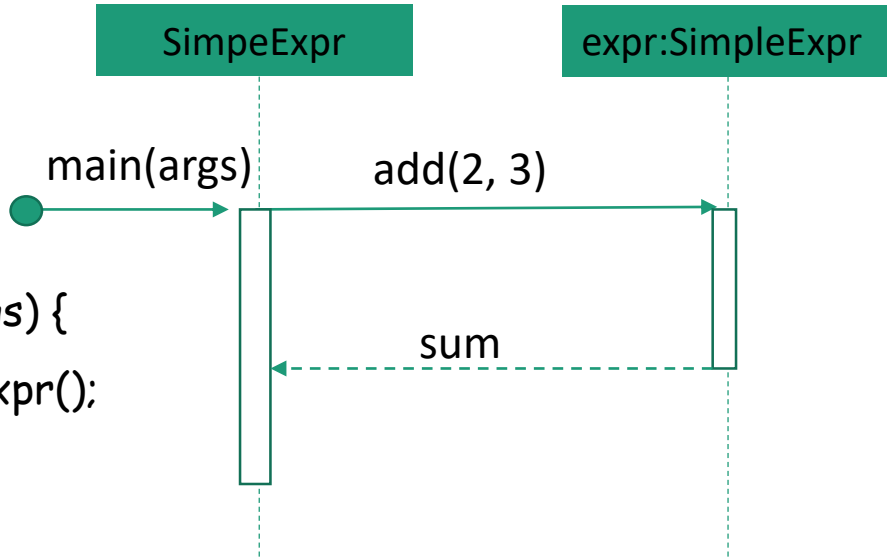
Return Value

- Method can return a single value to the caller
  - Return a primitive value (a value of a primitive type)
  - Return an "object", or strictly speaking, a value of the reference to an object

Return Primitive Value

- Return a primitive value

```
class SimpleExpr {
   int add(int x, int y) {
      return x+y;
   }
   public static void main(String[] args) {
      SimpleExpr expr = new SimpleExpr();
      int sum = expr.add(2, 3);
      Systme.out.println(sum);
   }
}
```

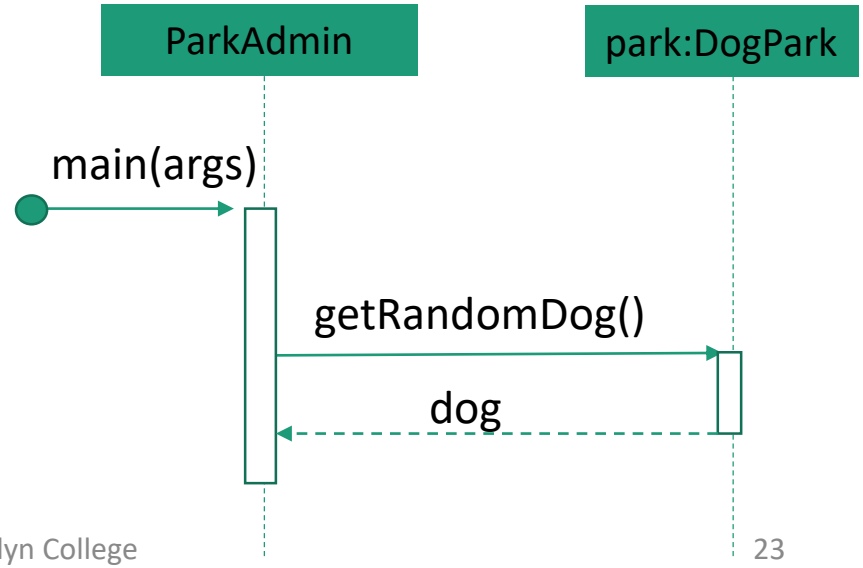


Return Reference

- Return an "object" (a value of the reference)

```
class DogPark {
    ……
    Dog getRandomDog() {
        int dogNo = (int) (Math.random() * 2);
        if (dogNo == 0)  {
            return buddy;
        } else {
            return millie;
        }
    }
}
```

```
class ParkAdmin { ……
    public static void main(String[] args) { ……
        Dog dog = park.getRandomDog();
        ……
```



Return Value

- What if I want to "return" multiple "values"?
- Consider a "list", e.g., an array
  - See getAllDogs() in the DogPark class
- Consider a "wrap" object

Encapsulation

- Control access to object states and behavior
  - Information-hiding
    - With data encapsulation, the details of its internal implementation remain hidden from the outside world.
  - Compare GoodDog.java and BadDog.java
    - Make fields private unless you have a good reason not to.
    - The getter and setter methods

Example: GoodDog and BadDog

```
class GoodDog {
  private String name;
  private int size;
  void setSize(int ns) { size = ns;  }
  void getSize() {  return size;   }
  public static void main(String[] args)
  {
     Dog buddy = new GoodDog();
     buddy.setSize(8);
  }
}
```

```
class BadDog {
   String name;
   int size;
   public static void main(String[] args)
   {
     Dog buddy = new BadDog();
     buddy.size = 8;
   }
}
```

Constructing Objects

- A constructor method is being called when you create an object for a class
- Java compiler provides a default constructor if you do not provide one explicitly
  - Default constructor is the constructor that has no parameters
  - Examples
    - Player p = new Player();
    - Dog d = new Dog();

Constructors with Parameters

- Constructors can have parameters
  - Commonly used to initialize the instance variables
- Example

```
class Dog {
   private String name; private int size;
   Dog(String nm, int sz) {
      name = nm; size = sz;
   }
   public static void main(String[] args) {
      Dog buddy = new Dog("buddy", 50);
   }
}
```

Questions

- Instance variables and methods
- Methods and arguments
- Simple constructors

Using IDE

- Integrated Development Environment
  - Editing, building, debugging, testing, refactoring, and other tools in one single easily accessible application
- Popular IDEs for Java
  - Eclipse, IntelliJ IDEA, NetBeans, and Others
- Eclipse is required IDE for this class
  - Choose "Eclipse for Java Developers"

Build Automation

- Building an application
  - How is an application compiled and built?
  - What does an application depend on?
  - How is the application be packaged?
- Examples
  - Eclipse JDT, Ant, Maven, Gradle
- Apache Maven is required for this class

Eclipse + Maven

- Latest version of Eclipse for Java Developers comes with Apache Maven
- Create new Maven project
  - Always select Maven project, e.g.,
    - File → New → Other… → Maven → Maven Project
- Import existing Maven project
  - File →Import → Maven → Existing Maven Project

Package

- Organize classes and methods into package (thinking C++ namespace)
  - Member without public and private, package accessible
  - More discussion about access control in C08

Search Solution Online

- What should I ask?
- Which one should I trust?
- How do I know I got the solution?
- Am I plagiarizing?

Assignments

- Practice Assignment
- CodeLab