Classes and objects

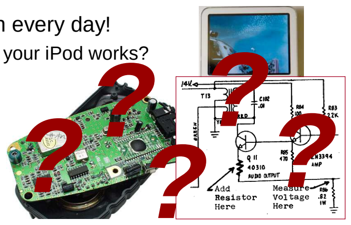
Readings: 8.1

Recall: Objects and classes

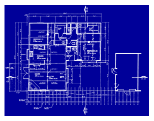
- **object:** An entity that contains data and behavior. We group objects into *classes*.
- **class:**
  - Basic building block of Java programs
    or
  - **Category or type of object**
- Classes we have seen so far: `String`, `Point`, `Scanner`, `DrawingPanel`, `Graphics`, `Color`, `Random`, `File`, `PrintStream`

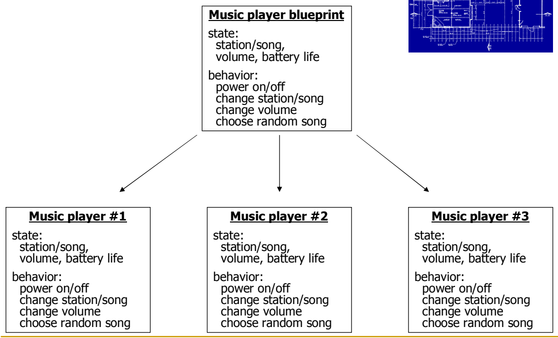
Big idea: Abstraction

- **abstraction**: A distancing between ideas and details.
  - How do objects provide a level of abstraction?
- You use abstraction every day!
  - Do YOU know how your iPod works?



Classes are like blueprints





Music player blueprint
state:
 station/song,
 volume, battery life
behavior:
 power on/off
 change station/song
 change volume
 choose random song

Music player #1
state:
 station/song,
 volume, battery life
behavior:
 power on/off
 change station/song
 change volume
 choose random song

Music player #2
state:
 station/song,
 volume, battery life
behavior:
 power on/off
 change station/song
 change volume
 choose random song

Music player #3
state:
 station/song,
 volume, battery life
behavior:
 power on/off
 change station/song
 change volume
 choose random song

Recall: `Point` object

- Constructing a `Point` object, general syntax:

```
Point <name> = new Point(<x>, <y>);
Point <name> = new Point();  // the origin, (0, 0)
```

- Examples:

```
Point p1 = new Point(5, -2);
Point p2 = new Point();
```

Recall: `Point` object

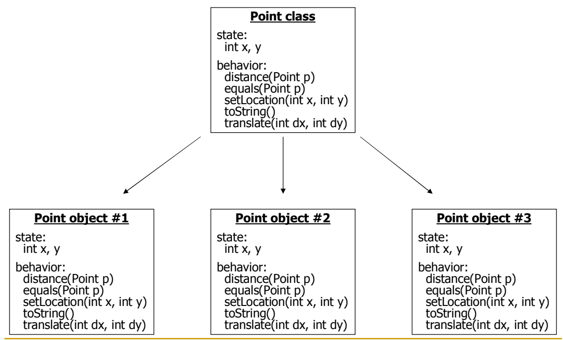
- Data stored in each `Point` object:

Field name	Description
x	the point's x-coordinate
y	the point's y-coordinate

- Useful methods in each `Point` object:

Method name	Description
`distance(`*p*`)`	how far away the point is from point *p*
`setLocation(`*x*`, `*y*`)`	sets the point's x and y to the given values
`translate(`*dx*`, `*dy*`)`	adjusts the point's x and y by the given amounts

`Point` class



Object state: fields

Readings: 8.2

`Point` class: Version 1

```
public class Point {
    int x;
    int y;
}
```

- Every object of type `Point` contains two integers.
- `Point` objects (so far) do not contain any behavior.
- Class declarations are saved in a file of the same name: `Point.java`

Fields

- **field**: A variable inside an object that represents part of its internal state.
  - Each object will have *its own copy* of the data fields we declare.
- Declaring a field, general syntax:
  <type> <name>;
  or
  <type> <name> = ***<value>***;   (with initialization)
- Example:

```
public class Student {
    String name;    // each student object has a
    double gpa;     // name and gpa data field
}
```

Accessing and modifying fields

- Accessing a data field, general syntax:
  <variable name>*.*<field name>
- Modifying a data field, general syntax:
  <variable name>*.*<field name> = ***<value>***;
- Example:

```
System.out.println("the x-coord is " + p1.x);  // access
p2.y = 13;                                      // modify
```

Client code

- The `Point` class is not an executable Java program. Why not?
  - It does not contain a `main` method.
- **client program**: Code that uses an object.

Client program: Version 1

```
public class PointMain {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 5;
        p1.y = 2;
        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");

        // move p2 and then print it again
        p2.x += 2;
        p2.y += 4;
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
    }
}
```

Output:

```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

Exercise

- Write a client program to produce the following output:

```
p1 is (7, 2)
p1's distance from origin = 7.280109889280518
p2 is (4, 3)
p2's distance from origin = 5.0
p1 is (18, 8)
p2 is (5, 10)
```

- Recall the formula to compute the distance between points (x_1, y_1) and (x_2, y_2) is: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Solution

```
public class PointProgram {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 7;
        p1.y = 2;
        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        double dist1 = Math.sqrt(p1.x * p1.x + p1.y * p1.y);
        System.out.println("p1's distance from origin = " + dist1);

        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
        double dist2 = Math.sqrt(p2.x * p2.x + p2.y * p2.y);
        System.out.println("p2's distance from origin = " + dist2);

        // move points and then print again
        p1.x += 11;
        p1.y += 6;
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        p2.x += 1;
        p2.y += 7;
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
    }
}
```

Object behavior: methods

Readings: 8.3

The case for methods

- How would we translate several points?

```
p1.x += 11;
p1.y += 6;

p2.x += 2;
p2.y += 4;

p3.x += 1;
p3.y += 7;
```

- What is unsettling about this code?

Attempt at eliminating redundancy

- Write a static method in the client code to translate points.

```
// Shifts the location of the given point.
public static void translate(Point p, int dx, int dy) {
    p.x += dx;
    p.y += dy;
}
```

- Example:

```
// move p2 and then print it again
translate(p2, 2, 4);
```

- Question: Why doesn't the method need to return the modified point?

Why is the static method solution bad?

- The call syntax doesn't match the way we're used to interacting with objects:

  ```
  translate(p2, 2, 4);
  ```

  We want something more like:

  ```
  p2.translate(2, 4);
  ```

- Every client code that wants to translate points would have to write their own static `translate` method.

Classes with behavior

- The whole point of writing classes is to put related state and behavior together.
- Point translation is closely related to the x/y data of the `Point` object, so it belongs in the `Point` class.

Instance methods

- **instance method**: A method inside an object that operates on that object.
- Declaring an object's method, general syntax:

  `public` ***<type> <name>*** `(`***<parameter(s)>***`) {`
      <statement(s)>`;`
  `}`

- How does this differ from previous methods?

Instance methods

- An object's instance methods can refer to its fields.

  ```
  public void translate(int dx, int dy) {
      x += dx;
      y += dy;
  }
  ```

- How does the `translate` method know which `x` and which `y` to modify?

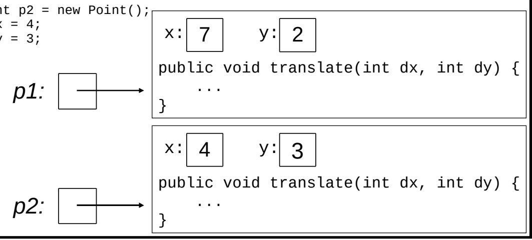
The implicit parameter

- Each instance method call happens on a particular object.
  - Example: `p1.translate(11, 6);`
- The code for an instance method has an implied knowledge of what object it is operating on.
- **implicit parameter**: The object on which an instance method is called.

`Point` object diagrams

- Think of each `Point` object as having its own copy of the `translate` method, which operates on that object's state:

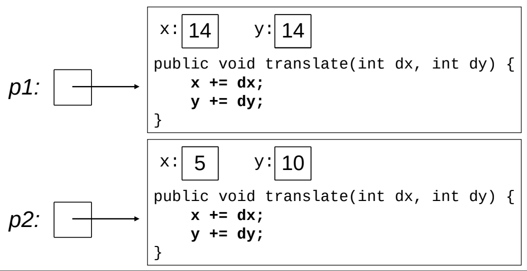
```
Point p1 = new Point();
p1.x = 7;
p1.y = 2;

Point p2 = new Point();
p2.x = 4;
p2.y = 3;
```



x: 7 y: 2
public void translate(int dx, int dy) {
...
}
p1:
x: 4 y: 3
public void translate(int dx, int dy) {
...
}
p2:

Tracing instance method calls

- What happens when the following calls are made?
  `p1.translate(11, 6);`
  `p2.translate(1, 7);`



```
public void translate(int dx, int dy) {
    x += dx;
    y += dy;
}
```

```
public void translate(int dx, int dy) {
    x += dx;
    y += dy;
}
```

`Point` class: Version 2

```
public class Point {
    int x;
    int y;

    // Changes the location of this Point object.
    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
```

Client program: Version 2

```
public class PointMain2 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.x = 5;
        p1.y = 2;
        Point p2 = new Point();
        p2.x = 4;
        p2.y = 3;

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
    }
}
```

Output:

```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

Errors in coding

- Why won't the following work in the client code?
  `translate(2, 4);`

```
PointMain2.java:16: cannot find symbol
symbol  : method translate(int,int)
location: class PointMain2
        translate(2, 4);
        ^
1 error
```

- In the client code, instance methods need to be called on an object, otherwise it is unknown what the `translate` method is (i.e. “cannot find symbol”), because it's not declared in the client code.

Exercises

- Write an instance method named `distanceFromOrigin` that computes and returns the distance between the current `Point` object and the origin, (0, 0).
- Write an instance method named `distance` that accepts a `Point` and computes the distance between it and the current `Point`.
- Write an instance method named `setLocation` that accepts x and y values and changes the `Point`'s location to be those values.
- **Modify the client code to use these new methods as appropriate.**

Solutions

```
public class Point {
    int x;
    int y;

    // Changes the location of this Point object.
    public void translate(int dx, int dy) {
        setLocation(x + dx, y + dy);
    }

    // Returns the distance from this Point object to the origin
    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }

    // Returns the distance from this Point object to the given point
    public double distance(Point other) {
        int dx = x - other.x;
        int dy = y - other.y;
        return Math.sqrt(dx * dx + dy * dy);
    }

    // Sets this Point object's location
    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }
}
```

Exercise

- Recall our client program that produces this output:

```
p1 is (7, 2)
p1's distance from origin = 7.280109889280518
p2 is (4, 3)
p2's distance from origin = 5.0
p1 is (18, 8)
p2 is (5, 10)
```

- Modify the program to use our new instance methods.
- Also add the following output to the program:

```
distance from p1 to p2 = 13.152946437965905
```

Solution

```
public class PointProgram {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point();
        p1.setLocation(7, 2);

        Point p2 = new Point();
        p2.setLocation(4, 3);

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        System.out.println("p1's distance from origin = " + p1.distanceFromOrigin());

        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
        System.out.println("p2's distance from origin = " + p2.distanceFromOrigin());

        // move points and then print again
        p1.translate(11, 6);
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        p2.translate(1, 7);
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");

        System.out.println("distance from p1 to p2 = " + p1.distance(p2));
    }
}
```

Test your understanding

- What is the significance of the `static` keyword?
  - instance method = NOT declared `static`
- Is `sqrt` in the Math class static? Why or why not?
  - Yes, because no object is needed to use `sqrt`.
- Is `nextInt` in the Scanner class static? Why or why not?
  - No, because you need a Scanner object to use `nextInt`.

Object initialization: constructors

Readings: 8.4

Initializing objects

- It is tedious to have to construct an object and assign values to all of its data fields manually.

```
Point p = new Point();
p.x = 3;
p.y = 8;                     // tedious
```

- We want something more like:

```
Point p = new Point(3, 8);   // better!
```

Constructor

- **constructor**: A special method that initializes the state of new objects as they are created.
- Constructor syntax:

  `public` ***<class name>*** `(`***<parameter(s)>***`) {`
      <statement(s)>`;`
  `}`

- How does this differ from previous methods?

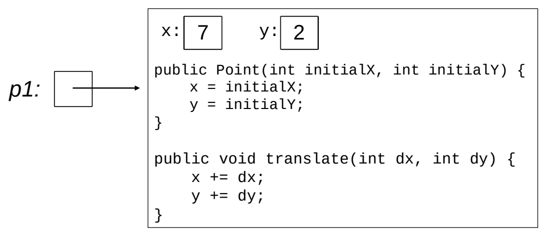
`Point` class: Version 3

```
public class Point {
    int x;
    int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    // Changes the location of this Point object.
    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
```

37

Tracing constructor calls

- What happens when the following call is made?
  `Point p1 = new Point(7, 2);`



```
public Point(int initialX, int initialY) {
    x = initialX;
    y = initialY;
}

public void translate(int dx, int dy) {
    x += dx;
    y += dy;
}
```

38

Uh oh!

- Our client code doesn't work anymore!

```
PointMain2.java:4: cannot find symbol
symbol  : constructor Point()
location: class Point
        Point p1 = new Point();
                   ^
PointMain2.java:7: cannot find symbol
symbol  : constructor Point()
location: class Point
        Point p2 = new Point();
                   ^
```

- Why did it work before?
  - If a class has no explicit constructor, Java gives it a *default constructor* with no parameters that sets all the object's fields to zero-equivalent values.

39

Client program: Version 3

```
public class PointMain3 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
    }
}
```

Output:

```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

40

Exercise

- Recall our client program that produces this output:

```
p1 is (7, 2)
p1's distance from origin = 7.280109889280518
p2 is (4, 3)
p2's distance from origin = 5.0
p1 is (18, 8)
p2 is (5, 10)
distance from p1 to p2 = 13.152946437965905
```

- Modify the program to use our new constructor.

41

Solution

```
public class PointProgram {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(7, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        System.out.println("p1's distance from origin = " + p1.distanceFromOrigin());

        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");
        System.out.println("p2's distance from origin = " + p2.distanceFromOrigin());

        // move points and then print again
        p1.translate(11, 6);
        System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
        p2.translate(1, 7);
        System.out.println("p2 is (" + p2.x + ", " + p2.y + ")");

        System.out.println("distance from p1 to p2 = " + p1.distance(p2));
    }
}
```

42

The keyword `this`

Readings: 8.7 (pg. 469 – 471)

43

`Point` class constructor

- What happens if the constructor has the following header?

```
public Point(int x, int y)
```

44

Variable shadowing

- **shadowed variable**: A field that is "covered up" by a local variable or parameter with the same name.
- Normally, it is illegal to have two variables in the same scope with the same name, but in this case (fields and local variables) it is allowed. Why?
- Otherwise, to avoid shadowing, we would always have to give the parameters different names:

```
public Point(int initialX, int initialY) {
    x = initialX;
    y = initialY;
}
```

45

Using the keyword `this`

- The `this` keyword is a reference to the implicit parameter (the object on which an instance method is being called).
- Usage of the `this` keyword, general syntax:
  - To refer to a field:
    `this.`***<field name>***
  - To refer to a method:
    `this.`***<method name>***`(`***<parameters>***`);`

46

It's like this and like that and like this and...

- The `this` keyword lets us use the same names and avoid shadowing:

```
public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

- When `this.` is present, the field is used.
- When `this.` is not present, the parameter is used.

47

Encapsulation

Readings: 8.5 (pg. 453 – 457)

48

Encapsulation

- **encapsulation**: Hiding the implementation details of an object from the clients of the object.
- Encapsulating objects provides *abstraction*, because we can use them without knowing how they work.

49

Implementing encapsulation

- Fields can be declared *private* to indicate that no code outside their own class can change them.
- Declaring a private field, general syntax:

```
private <type> <name>;
```

- Examples:

```
private int x;
private String name;
```

50

Private fields

- Once fields are private, client code cannot directly access them. The client receives an error such as:

```
PointMain3.java:8: x has private access in Point
System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
```

- How can the client program see what *x* is?

51

Accessors

- **accessor**: An instance method that provides information about the state of an object.
- Example:

```
public int getX() {
    return x;
}
```

- This gives clients "read-only" access to the object's fields.

52

Mutators

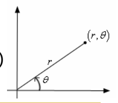
- **mutator**: An instance method that modifies the object's internal state.
- Example:

```
public void setX(int newX) {
    x = newX;
}
```

53

Benefits of encapsulation

- Provides a clean layer of abstraction between an object and its clients.
- Protects an object from unwanted access by clients.
  - Would you like any program to be able to modify the `BankAccount` object with your account information?
- Allows the class author to change the internal representation later if necessary.
  - Example: Changing the `Point` class to use polar coordinates (a radius *r* and an angle *θ* from the origin)



`Point` class: Version 4

```
public class Point {
    private int x;
    private int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    // Changes the location of this Point object.
    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
```

Client program: Version 4

```
public class PointMain4 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1 is (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("p2 is (" + p2.getX() + ", " + p2.getY() + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2 is (" + p2.getX() + ", " + p2.getY() + ")");
    }
}
```

`toString` or not `toString`

Readings: 8.6 (pg. 460 – 462)

Latest version of client code

```
public class PointMain4 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1 is (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("p2 is (" + p2.getX() + ", " + p2.getY() + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2 is (" + p2.getX() + ", " + p2.getY() + ")");
    }
}
```

- Any remaining redundancies?

Printing points

- Instead of

```
System.out.println("p1 is (" + p1.getX() + ", " + p1.getY() + ")");
```

- It would be nice to have something more like:

```
System.out.println("p1 is " + p1);
```

- What does this line currently do? Does it even compile?

  It will print: `p1 is Point@9e8c34`

`toString`

- When an object is printed or concatenated with a `String`, Java calls the object's `toString` method.

```
System.out.println("p1 is " + p1);
```

  is equivalent to:

```
System.out.println("p1 is " + p1.toString());
```

- Note: Every class has a `toString` method.

`toString`

- The default `toString` behavior is to return the class's name followed by gibberish (as far as you are concerned).
- You can replace the default behavior by defining a `toString` method in your class.

`toString` method syntax

- The `toString` method, general syntax:
```
public String toString() {
    <statement(s) that return a String>;
}
```
- NB: The method must have this exact name and signature (*i.e.*, `public String toString()`).
- Example:
```
// Returns a String representing this Point.
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

Object state

The Parent class

```
public class Parent {
    private int count;

    public Parent() {
        count = 0;
    }

    public String areWeThereYet() {
        count++;
        if (count >= 7) {
            return "NO!!!!  Now sit down and shut up, you ungrateful little brat!";
        } else if (count % 2 == 0) {
            return "We'll be there soon";
        } else {
            return "We're almost there";
        }
    }
}
```

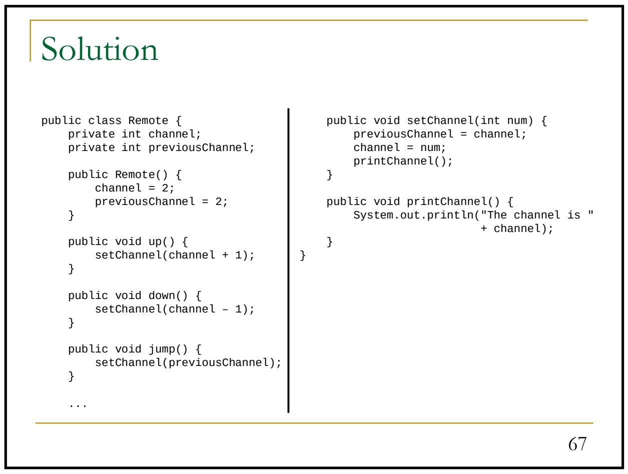
The Parent class: Version 2

```
public class Parent {
    private int count;
    private int threshold;

    public Parent(int threshold) {
        count = 0;
        this.threshold = threshold;
    }

    public String areWeThereYet() {
        count++;
        if (count >= threshold) {
            return "NO!!!!  Now sit down and shut up, you ungrateful little brat!";
        } else if (count % 2 == 0) {
            return "We'll be there soon";
        } else {
            return "We're almost there";
        }
    }
}
```

Exercise

- Write a class `Remote` that implements a TV remote control with a “jump” button. The remote keeps track of the TV channel. When the user presses “jump”, the channel is set to the previous channel.

The remote should have the following methods:
- `up()`: sets the channel to be the next one up
- `down()`: sets the channel to be the next one down
- `setChannel(int)`: sets the channel to an arbitrary channel
- `jump()`: sets the channel to the previous channel



Solution

```
public class Remote {
    private int channel;
    private int previousChannel;

    public Remote() {
        channel = 2;
        previousChannel = 2;
    }

    public void up() {
        setChannel(channel + 1);
    }

    public void down() {
        setChannel(channel – 1);
    }

    public void jump() {
        setChannel(previousChannel);
    }

    ...
```

```
    public void setChannel(int num) {
        previousChannel = channel;
        channel = num;
        printChannel();
    }

    public void printChannel() {
        System.out.println("The channel is "
                           + channel);
    }
}
```

67