

# Data Types

- What Java types are you familiar with?
  - Primitive types?
  - Reference types?
- What does a type tell us about a variable?
- Java classes can be used to define new types

# Examples

- String—string of characters
  - length, charAt, indexOf, substring, etc.
- Question—multiple choice question
  - getQuestion, getOption, getAnswer, etc.
- PrintWriter—output stream
  - print, println
- BufferedReader—input stream
  - readLine

## A New Type

- Person
  - values: (name , age)
  - operations:
    - **void** setName(String)
    - **void** setAge(int)
    - String getName()
    - **int** getAge()

## Using The New Type

```
// create a new Person object
Person p = new Person();
p.setName("Al Dente"); // sets p's name
p.setAge(22);          // sets p's age

// Output friendly message
System.out.println("Hi, " + p.getName() + "!");
```

- What is the output?

## Another New Type

- **Rectangle**
  - values: (width, height, x, y)
  - operations:
    - **void** setWidth(**int**)
    - **void** setHeight(**int**)
    - **void** setX(**int**)
    - **void** setY(**int**)

cont.

## Another New Type cont.

- **int** getWidth()
- **int** getHeight()
- **int** getX()
- **int** getY()
- **int** getPerimeter()
- **int** getArea()
- **void** translate(**int**, **int**)
- **void** draw()

## Example Trace

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle();

r1.setWidth(10);
r1.setHeight(5);
r2.setWidth(17);
r2.setHeight(24);
System.out.println("r1: width = " + r1.getWidth() +
    ", height = " + r1.getHeight());
System.out.println("r2: width = " + r2.getWidth() +
    ", height = " + r2.getHeight());
```

## Example Client

- Write a piece of code that creates a Rectangle object, asks the user for the rectangle's width and height and sets these values in the created object, and finally computes and outputs the rectangle's perimeter and area.
- Use the Rectangle methods to handle all computations.

# Rectangle Client

# Defining New Types

- In Java, classes are used to define new types
- In a class definition, we need to provide two things:
  - data fields (a.k.a. instance variables) to represent the values of the objects of the new type
  - instance methods to provide operations to manipulate objects of the new type

## First Example: class Person

```
public class Person {  
    // data fields (instance variables)  
    private String name;  
    private int age;  
  
    // instance methods (no static keyword!!)  
    public void setName(String newName) {...}  
    public void setAge(int newAge) {...}  
    public String getName() {...}  
    public int getAge() {...}  
}
```

## First Example (cont.)

- How do we implement (write) the instance methods?
- Instance methods have access to the instance variables, so they can modify them directly, e.g.,

```
public void setName(String newName) {  
    name = newName;  
}
```

## Your Turn

- Write the body of the remaining three operations:
  - setAge
  - getName
  - getAge

## Completed class Person

```
public class Person {  
    // data fields (instance variables)  
    private String name;  
    private int age;  
  
    // instance methods (no static keyword!!)  
    public void setName(String newName) {  
        name = newName;  
    }  
    public void setAge(int newAge) {  
    }  
    public String getName() {  
    }  
    public int getAge() {  
    }  
}
```

## Second Example: class Rectangle

```
public class Rectangle {  
    // data fields—what goes here?  
  
    // instance methods—what goes here?  
  
}
```

## Rectangle Data Fields

```
public class Rectangle {  
    // data fields  
    private int width; // rectangle width  
    private int height; // rectangle height  
    private int x; // x coordinate of left  
    // bottom corner  
    private int y; // y coordinate of left  
    // bottom corner
```

## Rectangle Instance Methods

```
public void setWidth(int newWidth) {
    width = newWidth;
}

public void setHeight(int newHeight) {

}

public void setX(int newX) {

}

public void setY(int newY) {

}
```

## Rectangle Inst. Methods cont.

```
public int getWidth() {
    return width;
}

public int getHeight() {

}

public int getX() {

}

public int getY() {

}
```

## Rectangle Inst. Methods cont.

```
public int getPerimeter() {  
  
}  
public int getArea() {  
  
}  
public void translate(int deltaX, int deltaY) {  
  
}
```

## Rectangle Inst. Methods cont.

```
public void draw() {  
  
  
}  
} // end of class Rectangle
```

## Your Turn: class Student

- Student
  - values: (name, score1, score2, score3)
  - operations:
    - String getName() – returns the student name
    - int getScore (int i) – returns the i-th score ( $1 \leq i \leq 3$ )
    - void setScore (int i, int x) – sets the i-th score to x ( $1 \leq i \leq 3$  and  $x \geq 0$ )

## class Student

```
public class Student {  
    // data fields—what goes here?  
  
    // instance methods—what goes here?  
  
}
```

## Student Data Fields

```
public class Student {  
    // data fields
```

## Student Instance Methods

```
public String getName() {  
    }  
  
public int getScore(int i) {  
  
  
    }  
}
```



## Initial Values of Objects

- What is the initial value of a Person object *p* after:

```
Person p = new Person();
```

- What about the value of a Rectangle object *r* after:

```
Rectangle r = new Rectangle();
```

## Automatic Initialization

- By default, the instance variables are initialized as follows:
  - 0, for all numeric types
  - false, for boolean
  - null, for reference types
- Note that local variables (inside methods) do not get initialized automatically—the compiler will complain about it

# Constructors

- Before the instance variables even come into existence and are initialized, the object needs to be created
- *Constructors* are the special operations that take care of that
- If no constructor is defined explicitly, Java defines a *default* constructor which simply creates the object and initializes the instance variables to their default values

# Constructors cont.

- A constructor is a special kind of method
- The constructor name is the name of the class
- Like any other method constructors can have zero or more parameters
- Constructors are declared without a return type or void
- Constructors are invoked after the **new** operator when creating objects

## Constructor Example

```
public class Person {
    // data fields (instance variables)
    private String name;
    private int age;

    public Person () {
        this.name = null; // this is not necessary
        this.age = 0;     // this is not necessary
    }

    public Person (String name) {
        this.name = name;
        this.age = 0;    // this is not necessary
    }

    // instance methods
    ...
}
```

## Constructor Usage

- Given the second constructor on the previous slide, we can create a Person object as:

```
Scanner in = new Scanner(System.in);
System.out.print("What's your name? ");
String myName = in.nextLine();
Person p = new Person(myName);
```

## Another Constructor Example

```
public class Student {
    // data fields (instance variables)
    private String name;
    private int score1;
    private int score2;
    private int score3;

    public Student (String name) {
        this.name = name;
        // no need to set the scores to 0
    }

    // instance methods
    ...
}
```

## Your Turn

- Define another constructor for the Student class which takes a String and three integer parameters and initializes the Student instance variables with the given values.
- Then declare a student variable and initialize it with a Student object with name “Al Fresco”, and scores 100, 200, 300.

# New Student Constructor