

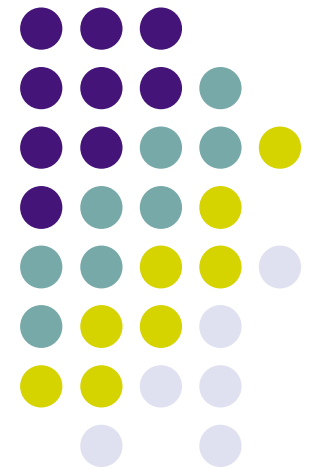


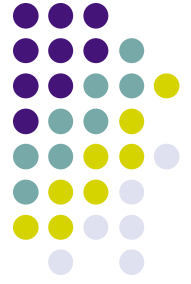
Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120 – Fall 2006

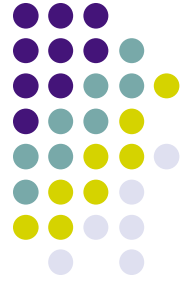
Lecture Unit 2 - Using Objects





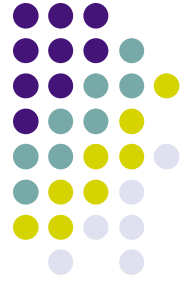
Lecture Outline

- Working with types and variables
- Classes and objects
- Methods
- Parameters and return values
- Constructing and using objects
- API documentation



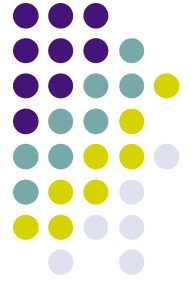
Types

- Every value (piece of data) has a *type*
 - "Hello World" : String
 - System.out : PrintStream
 - 13 : int
- Type determines what can be done with the values
 - Can call `println` on any `PrintStream` object
 - Can compute sum/product of any `int(eger)s`



Variables

- To store values for use at a later time
- A *variable* is a storage location in memory with
 - Type (what type of data can be stored)
 - Name (how you refer to the data)
 - Contents (the actual data)
- Variables must be *declared* before use:
 - ```
String greeting = "Hello, World!";
PrintStream printer = System.out;
...
printer.println(greeting);
```



# Syntax: Variable Declaration

*typeName* *variableName* = *value*;

or

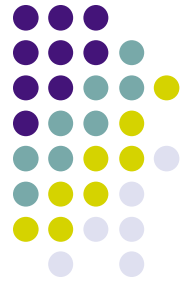
*typeName* *variableName*;

## **Example:**

```
String greeting = "Hello, Dave!";
int x;
```

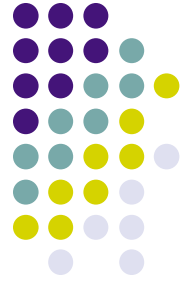
## **Purpose:**

To define a new variable of a particular type and optionally supply an initial value



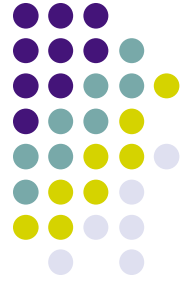
# Identifiers (Names)

- Identifier: name of a variable, method, or class
  - Case sensitive: greeting and Greeting different
- Rules
  - Made up of letters, digits, underscore \_
    - No other symbols allowed, including spaces
  - May not start with a digit
  - May not be a reserved word, like 'public'
- Conventions
  - Variable and method names start with lowercase
  - Class names start with uppercase letter
  - Use 'camelCase' names
- Conventions are useful for other people to be able to easily read and understand your code



# Assignment Operator

- Use **=** (assignment operator) to change value of an existing variable
- Note: **=** symbol does not refer to equality in Java
  - `12 = 12;`
- Error to use variable that does not have value assigned



# Syntax: Assignment

*variableName = value;*

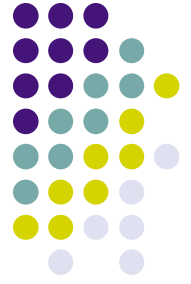
## **Example:**

LuckyNumber = 12;

## **Purpose:**

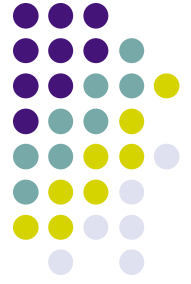
To assign a new value to a previously defined variable





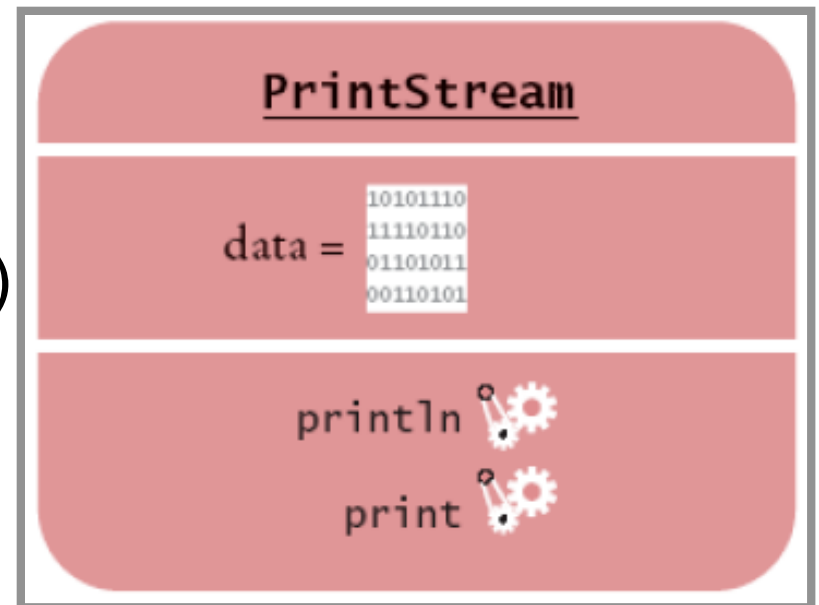
# Objects

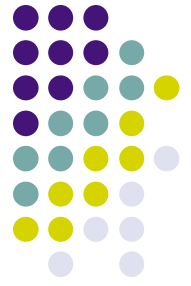
- ‘Things’ that you can manipulate in your Java programs
  - Represent entities in real world: bank accounts, employee records, graphical shapes, computer game player
- Often don’t know detailed internal structure (data) of objects
  - Can still manipulate objects by calling *methods*



# Classes

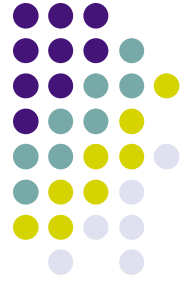
- Every object belongs to a *class*
  - System.out object (representing terminal output window) belongs to PrintStream class
- Classes are blueprints for creating and using objects
  - Define internal data (*fields*)
  - Define operations (*methods*)





# Methods

- Sequence of instructions to carry out some operation
  - Usually accesses internal data of an object
  - Every method has a name
  - May take some input(s) and return some output
- Objects belonging to the same class all support the same methods (operations)
- To get a method to carry out its operation, you *call* or *invoke* the method



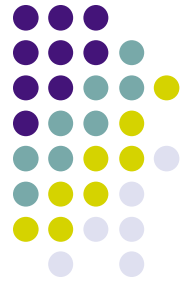
# Object/Method Examples

- `System.out`
  - `print()`
  - `println()`
  
- `"Hello World"`
  - `length()`
  - `toUpperCase()`

# Class/Method/Object Summary

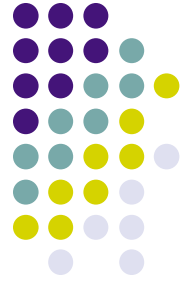


- Every object belongs to a class
- Class defines methods for its objects
  - These form the *public interface* of the class
- Class also defines data stored inside objects
  - These form the *private implementation*
  - Details (most often) hidden from other programmers using your objects and methods



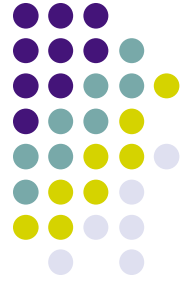
# Method Parameters

- Input provided to a method to give details about operation to be performed
  - `println` method takes a string parameter (input) to tell what to print out on the screen
- `System.out.println( "Hello, World!" );`
  - "Hello World" is an *explicit parameter*
  - Object on which method is called is also an *implicit parameter*
  - Length method of String class needs no explicit parameters



# Return Values

- Result of a method's computation
- Length method returns a value: the number of characters in the string
- Return values can be
  - Stored in a variable
  - Used as parameter of another method
- `String river2 = river.replace("issipp", "our")`
- `greeting.replace("World", "Dave").length()`



# Method Definition Headers

- String class:

- `public int length()`
- `public String replace(String target, String replacement)`

• Return value types

• Parameter types

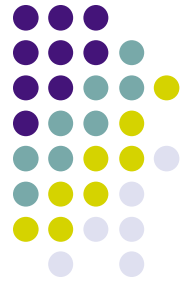
- PrintStream class:

- `public void println(String output)`
- `public void println(int output)`

• “Void” method returns no value

- Overloaded methods: two methods with same name but different parameters





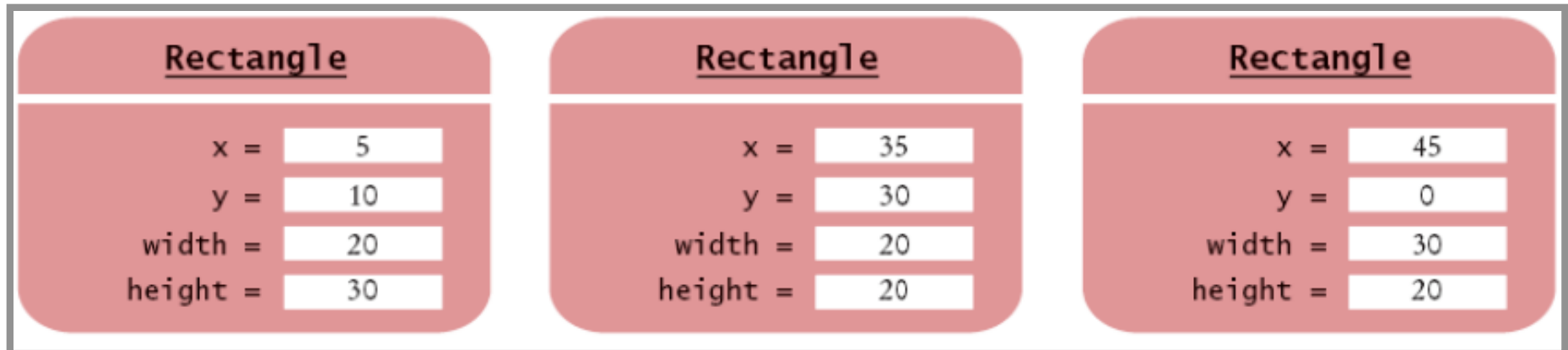
# Aside: Number Types

- *Integers*: whole numbers
  - 14   -7   13000
  - Java type: `int` (or `short`, or `long`)
- *Floating-point*: numbers with fractional parts
  - 1.3   0.00013   -1300.0
  - Java type: `double` (or `float`)
- Numbers are of *primitive types*, not objects
  - Number types have no methods
  - Numbers can be combined using arithmetic operators `+*-/`

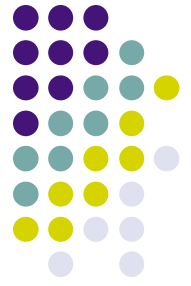


# Rectangle Objects

- Objects of type `Rectangle` *describe* rectangular shapes
  - `Rectangle` class is predefined in Java library



- Understand the distinction: `Rectangle` object is block of memory storing some data
  - In programmer's mind, object describes a geometric figure



# Constructing Objects

- To 'make' a new rectangle:

```
new Rectangle(5, 10, 20, 30)
```

- The `new` operator takes

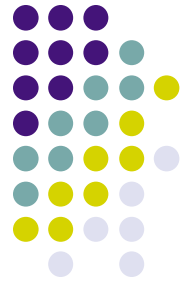
- name of a class (Rectangle)
- additional parameters required to construct a new object of that class (x, y, width, height)

- `new` operator returns the newly constructed object

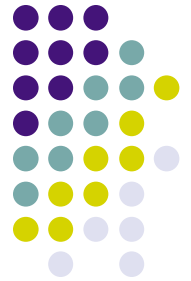
- Usually one stores the result in a variable:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

# Constructing Objects (cont.)



- Many classes allow construction of objects in multiple ways
  - `new Rectangle()`
  - All parameters are taken as being 0 (zero)



# Syntax: Object Construction

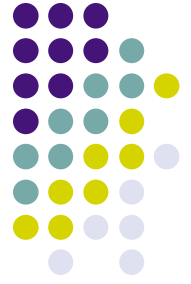
```
new ClassName(parameters);
```

## Examples:

```
new Rectangle(5, 10, 20, 30)
new Rectangle()
```

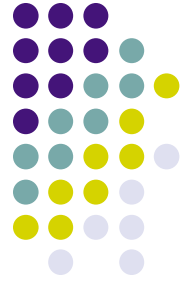
## Purpose:

To construct a new object, initialize it with the construction parameters, and return a reference to the constructed object



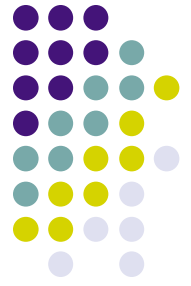
# Accessor/Mutator Methods

- *Accessor* -
  - method that accesses object and returns some information about it  
`double width = box.getWidth();`
- *Mutator* -
  - method that modifies the state of the object  
`box.translate(15, 25);`
  - Given `box` object of unknown dimensions, how do you translate it so the x-coordinate becomes 0?



# Writing a Test Program

- Provide a new class
- Define a main method
- Inside the main method, construct object(s)
  - Rectangle  $x=5$ ,  $y=10$ ,  $width=20$ ,  $height=30$
- Apply object methods
  - Move rectangle 15 pixels horizontally, 25 vertically
- Display results



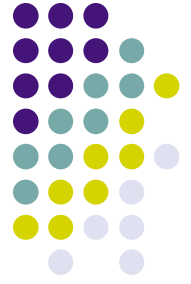
# Importing Packages

- Java classes are grouped into *packages*
  - Packages are grouped into a *library*
- To use class(es) defined in another package, you must *import* them at the beginning of your program file:

```
import java.awt.Rectangle;
```

  - 'awt' = Abstract Windowing Toolkit
- System and String classes are in `java.lang` package - automatically imported





# Syntax: Importing Classes

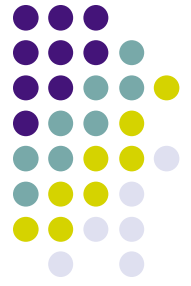
```
import packageName.ClassName;
```

## **Examples:**

```
import java.awt.Rectangle;
```

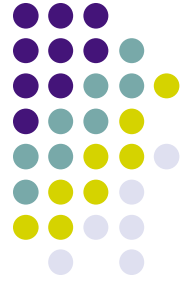
## **Purpose:**

To import a class from a package for use in a program



# Writing and Testing Code

- Using DrJava...
  
- Write program that
  - constructs two rectangle objects with arbitrary position/size
  - constructs a third rectangle with
    - top-left corner halfway between top-left corners of original two
    - Width and height the average of the original two



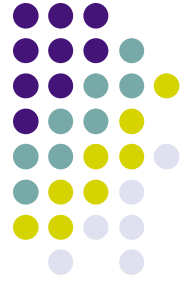
# Object References

- (Section 2.10)
- Primitive type variables store actual values
- Object variables store *references* to objects
  - Multiple object variables can refer to same object

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
```

```
int luckyNum = 13;
int luckyNum2 = luckyNum;
```

```
// translate, toUpperCase...
```

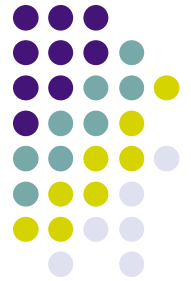


# API Documentation

- API = Application Programming Interface
- Documentation lists classes and methods in Java library
  - <http://java.sun.com/j2se/1.5/docs/api/>
- Not possible to memorize entire API
  - Use online documentation, or download it to your computer
- Self-Check 22, 23 (pg. 52)

# Programming

- Exercise P2.10
- Project 2.1



# Random Fact 2.1: Mainframes

