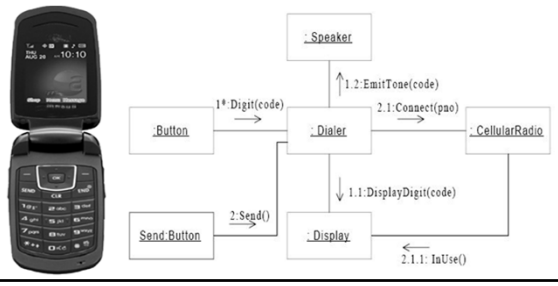
CSIS 3103

Object Oriented Programming with Java

Object-Oriented Programming

Objects are the basic run-time entities

View a program as a system of interacting objects



Object-oriented programming is about modeling and simulation

- The idea is to create a *model* of the part of the world (real or imaginary)
- Each object has it's own things it knows and things it can do

Java is object-oriented

- *Almost* everything in Java is an *object*
- Objects *know* things and can *do* things
  - *Variables* store what they *know* (data)
  - *Methods* provide the actions they can do
- Objects are instances of a given class in Java.

Concrete Data Types

A *data type* specifies:

- A *set of values*
- The *operations* that may be performed on those values

Primitive Data Types

Java has eight *primitive data types*:

```
boolean
byte
char
short
int
long
float
double
```

Example: `int` is a data type that specifies:
the set values {–2147483648, …, 2147483647}
the operations +, –, *, /, %

Representation of Primitive Types

A box (memory location) represents a variable

A *declaration* builds a box

An *assignment* fills a box with a value

```
int n = 0;
```
n ~~0~~ 5

```
n = 5;
```

```
int x = n;
```
x 5

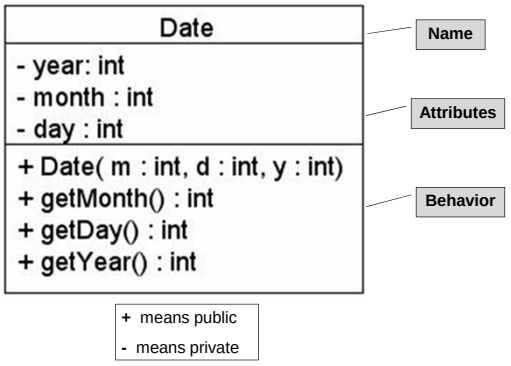
Java Classes

A *class* defines
- a data type (can be used in declarations, etc.)
- a set of objects
- a way to construct objects
- the operations to modify and observe objects

Unified Modeling Language (UML)

An object oriented modeling language used to represent classes, objects, and relationships in a software system

Class Diagram



Instance Variables (fields)

A *variable* is identified by its attributes:
- name
- type
- visibility (public, private, protected)

Example

```
private int year;
```

visibility | type | name

Variables store the *state* of an object

Methods

A *method* is identified by its attributes:
- name
- visibility (public, private, protected)
- return value type
- parameters

Example

```
public void setYear(int year)
```

visibility | return type | name | parameter

Methods provide the *behavior* of an object

Constructors

A class normally has *constructors* to initialize the state of new objects

- All constructors use the name of the class
- Constructors do not have a return type
- If a class has more than one constructor, Java selects the right one by matching the constructor's *signature* (parameter types, number, order)

Objects and Reference Types

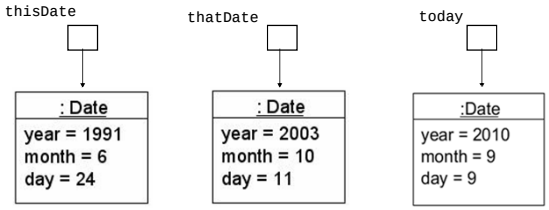
- All non-primitive types in Java are *reference* types
- An *object* is an instance of a non-primitive type
- A primitive variable stores a value
- A reference variable stores a reference to an object (its memory address)

Build Objects with the **new** operator

1. Builds a box for the object
2. Calls the appropriate constructor to initialize the contents of the object
3. Returns a reference to the object

Object Examples

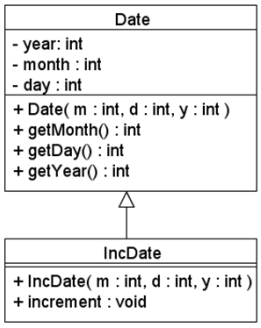
```
Date thisDate =  new Date (6, 24, 1991);
Date thatDate = new Date (10, 11, 2003);
Date today = new Date (9, 9, 2010);
```



Inheritance

1. Allows programmers to create a new class that is a specialization of an existing class.
2. The new class is called a *subclass* of the existing class.
   The existing class is the *superclass* of the new class.
3. The subclass *inherits* all the public and protected parts of its parent.

Class Diagram for IncDate (a class that can increment a Date)



Examples

```
Date d1 = new Date(9, 8, 2006);
IncDate d2 = new IncDate(12, 31, 2004);

d1.getYear()
d2.getYear()
d1.increment(); // error
d2.increment();
d1 = d2;
d2 = d1; // error
```

Public vs. Private

- Methods and variables in a class may be *public* or *private*
- Public things can be accessed anywhere
  - Variables can be read and assigned
  - Methods can be called
- Private things can only be accessed by methods defined within the same class

What Should Be Private?

Why are `month`, `day`, `year` in `Date` "private"?

- hides unnecessary details from clients
- "clients don't need to know about this…"
- keeps clients from relying on it
- allows for future changes in the code
- supports keeping invariants true
  `month` should be `1..12` only

The Meaning of ==

For primitive types, == is true if the stored values are identical.

For reference types, == is true if the references are to the same object (or both are `null`).

Classes usually should include an `equals` method for comparing.

Examples

```
int a = 6;
int b = 10;  // a == b is false
b -= 4;      // a == b is true
Date d1 = new Date(9, 10, 2010);
Date d2 = new Date(9, 10, 2010);
   // d1 == d2 is false
d1 = d2;
   // Now d1 == d2 is true
```