Defining classes and objects in Java

Madhavan Mukund

`https://www.cmi.ac.in/~madhavan`

Programming Concepts using Java

Week 2

Classes and objects

- A class is a template for an encapsulated type
- An object is an instance of a class
- How do we create objects?
- How are objects initialized?

Defining a class

- Definition block using `class`, with class name
    - Modifier `public` to indicate visibility
    - Java allows `public` to be omitted
    - Default visibility is public to `package`
    - Packages are administrative units of code
    - All classes defined in same directory form part of same package

```
public class Date {

  private int day, month, year;

  ...

}
```

Defining a class

- Definition block using `class`, with class name
  - Modifier `public` to indicate visibility
  - Java allows `public` to be omitted
  - Default visibility is public to `package`
  - Packages are administrative units of code
  - All classes defined in same directory form part of same package
- Instance variables
  - Each concrete object of type `Date` will have local copies of `date`, `month`, `year`
  - These are marked `private`
  - Can also have `public` instance variables, but breaks encapsulation

```
public class Date {

  private int day, month, year;

  ...

}
```

Creating objects

- Declare type using class name
- `new` creates a new object
    - How do we set the instance variables?

```
public void UseDate() {
  Date d;
  d = new Date();
  ...
}
```

Creating objects

- Declare type using class name
- `new` creates a new object
  - How do we set the instance variables?
- Can add methods to update values
  - `this` is a reference to current object

```
public void UseDate() {
  Date d;
  d = new Date();
  ...
}
```

```
public class Date {
  private int day, month, year;

  public void setDate(int d, int m,
                      int y){
    this.day = d;
    this.month = m;
    this.year = y;
  }
}
```

Creating objects

- Declare type using class name
- `new` creates a new object
    - How do we set the instance variables?
- Can add methods to update values
    - `this` is a reference to current object
    - Can omit `this` if reference is unambiguous

```
public void UseDate() {
  Date d;
  d = new Date();
  ...
}
```

```
public class Date {
  private int day, month, year;

  public void setDate(int d, int m,
                      int y){
    day = d;
    month = m;
    year = y;
  }
}
```

Creating objects

- Declare type using class name
- new creates a new object
  - How do we set the instance variables?
- Can add methods to update values
  - this is a reference to current object
  - Can omit this if reference is unambiguous
- What if we want to check the values?
  - Methods to read and report values

```
public class Date {
  ...

  public int getDay(){
    return(day);
  }

  public int getMonth(){
    return(month);
  }

  public int getYear(){
    return(year);
  }

}
```

Creating objects

- Declare type using class name
- `new` creates a new object
  - How do we set the instance variables?
- Can add methods to update values
  - `this` is a reference to current object
  - Can omit `this` if reference is unambiguous
- What if we want to check the values?
  - Methods to read and report values
- Accessor and Mutator methods

```
public class Date {
  ...

  public int getDay(){
    return(day);
  }

  public int getMonth(){
    return(month);
  }

  public int getYear(){
    return(year);
  }

}
```

Initializing objects

- Would be good to set up an object when we create it
  - Combine `new Date()` and `setDate()`

Initializing objects

- Would be good to set up an object when we create it
  - Combine `new Date()` and `setDate()`
- Constructors — special functions called when an object is created
  - Function with the same name as the class
  - `d = new Date(13,8,2015);`

```
public class Date {
  private int day, month, year;

  public Date(int d, int m, int y){
    day = d;
    month = m;
    year = y;
  }
}
```

Initializing objects

- Would be good to set up an object when we create it
  - Combine `new Date()` and `setDate()`
- Constructors — special functions called when an object is created
  - Function with the same name as the class
  - `d = new Date(13,8,2015);`
- Constructors with different signatures
  - `d = new Date(13,8);` sets `year` to `2021`
  - Java allows function overloading — same name, different signatures
    - Python: default (optional) arguments, no overloading

```
public class Date {
  private int day, month, year;

  public Date(int d, int m, int y){
    day = d;
    month = m;
    year = y;
  }

  public Date(int d, int m){
    day = d;
    month = m;
    year = 2021;
  }
}
```

Constructors . . .

- A later constructor can call an earlier one using `this`

```
public class Date {
  private int day, month, year;

  public Date(int d, int m, int y){
    day = d;
    month = m;
    year = y;
  }

  public Date(int d, int m){
    this(d,m,2021);
  }
}
```

Constructors . . .

- A later constructor can call an earlier one using `this`
- If no constructor is defined, Java provides a default constructor with empty arguments
  - `new Date()` would implicitly invoke this
  - Sets instance variables to sensible defaults
  - For instance, `int` variables set to `0`
  - Only valid if *no* constructor is defined
  - Otherwise need an explicit constructor without arguments

```
public class Date {
  private int day, month, year;

  public Date(int d, int m, int y){
    day = d;
    month = m;
    year = y;
  }

  public Date(int d, int m){
    this(d,m,2021);
  }
}
```

Copy constructors

- Create a new object from an existing one

```
public class Date {
  private int day, month, year;

  public Date(Date d){
    this.day = d.day;
    this.month = d.month;
    this.year = d.year;
  }
}
```

Copy constructors

- Create a new object from an existing one
- Copy constructor takes an object of the same type as argument
  - Copies the instance variables
  - Use object name to disambiguate which instance variables we are talking about
  - Note that private instance variables of argument are visible

```
public class Date {
  private int day, month, year;

  public Date(Date d){
    this.day = d.day;
    this.month = d.month;
    this.year = d.year;
  }
}


public void UseDate() {
  Date d1,d2;
  d1 = new Date(12,4,1954);
  d2 = new.Date(d1);
}
```

Copy constructors

- Create a new object from an existing one
- Copy constructor takes an object of the same type as argument
  - Copies the instance variables
  - Use object name to disambiguate which instance variables we are talking about
  - Note that private instance variables of argument are visible
- Shallow copy vs deep copy
  - Want new object to be disjoint from old one
  - If instance variable are objects, we may end up aliasing rather than copying
  - Discuss later — cloning objects

```
public class Date {
  private int day, month, year;

  public Date(Date d){
    this.day = d.day;
    this.month = d.month;
    this.year = d.year;
  }
}


public void UseDate() {
  Date d1,d2;
  d1 = new Date(12,4,1954);
  d2 = new.Date(d1);
}
```

Summary

- A class defines a type
- Typically, instance variables are private, available through accessor and mutator methods
- We declare variables using the class name as type
- Use `new` to create an object
- Constructor is called implicitly to set up an object
  - Multiple constructors — overloading
  - Reuse — one constructor can call another
  - Default constructor, if none is defined
  - Copy constructor — make a copy of an existing object