Computer Science 227

Objects and the Static Modifier

Brian Nakayama[1]

[1] Department of Computer Science, Iowa State University, Ames, IA 50010, USA

Jan 20th, 2017

Method Review

- A method is like a function. For example:

$$f(x) = x + 7$$

- When $x = 2$, $f(x) =$

Method Review

- A method is like a function. For example:

$$f(x) = x + 7$$

- When $x = 2$, $f(x) = 9$. In java this would look like:

Method Review

- A method is like a function. For example:

$$f(x) = x + 7$$

- When $x = 2$, $f(x) = 9$. In java this would look like:

```
public static int f(int x){
     return x + 7;
}
```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double
```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(
```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

```
public static void
```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

```
public static void printStuff(int number, String s)
```

- A method that checks if a number is prime:

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

  ```
  public static void printStuff(int number, String s)
  ```

- A method that checks if a number is prime:

  ```
  public static boolean
  ```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

  ```
  public static void printStuff(int number, String s)
  ```

- A method that checks if a number is prime:

  ```
  public static boolean isPrime(int number)
  ```

- A method that adds an explanation mark to a String:

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

  ```
  public static void printStuff(int number, String s)
  ```

- A method that checks if a number is prime:

  ```
  public static boolean isPrime(int number)
  ```

- A method that adds an explanation mark to a String:

  ```
  public static String
  ```

Making Methods

How would you declare the following methods:

- A method that calculates interest for an arbitrary amount of money and time:

```
public static double calcInterest(double money, double time)
```

- A method that prints an integer number of a string to the console:

  ```
  public static void printStuff(int number, String s)
  ```

- A method that checks if a number is prime:

  ```
  public static boolean isPrime(int number)
  ```

- A method that adds an explanation mark to a String:

  ```
  public static String soExcited(String s)
  ```

Fill in the Code

```
public __1__ CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static __2__ encrypt(__3__, ____4__){

        //Check if c is an ASCII character
        __5__ ('a' <= c __6__ __7__ __8__ 1 __9__ __10__ __11__){
            __12__ newKey __13__ (c - 'a' + key) % 26;
            newKey __14__ 'a'__15__
            return __16__;
        } __17__ {
            //If c is not a correct char, don't encrypt.
            __18__
        __19__
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static _2_ encrypt(_3_, _____4_){

        //Check if c is an ASCII character
        _5_ ('a' <= c _6_ __7___ _8_ 1 __9___ _10_ ___11____){
            _12_ newKey _13_ (c - 'a' + key) % 26;
            newKey _14_ 'a'_15_
            return __16__;
        } _17_ {
            //If c is not a correct char, don't encrypt.
            __18____
        _19_
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(  3   , _____4 ){

        //Check if c is an ASCII character
        _5_ ('a' <= c _6_ __7____ _8_ 1 __9___ _10_ ___11____){
            _12_ newKey _13_ (c - 'a' + key) % 26;
            newKey _14_ 'a'_15_
            return __16__;
        } _17_ {
            //If c is not a correct char, don't encrypt.
            __18____
        _19_
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, ____4____){

        //Check if c is an ASCII character
        _5_ ('a' <= c _6_ __7___ _8_ 1 __9___ _10_ ___11____){
            _12_ newKey _13_ (c - 'a' + key) % 26;
            newKey _14_ 'a'_15_
            return __16__;
        } _17_ {
            //If c is not a correct char, don't encrypt.
            __18____
        _19_
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        _5_ ('a' <= c _6_ __7___ _8_ 1 __9___ _10_ ___11____){
            _12_ newKey _13_ (c - 'a' + key) % 26;
            newKey _14_ 'a'_15_
            return __16__;
        } _17_ {
            //If c is not a correct char, don't encrypt.
            __18____
        _19_
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c  6    7    8  1   9    10    11    ){
            12  newKey  13  (c - 'a' + key) % 26;
            newKey  14  'a' 15
            return   16  ;
        }  17  {
            //If c is not a correct char, don't encrypt.
              18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && __7__ __8__ 1 __9__ __10__ __11__){
            __12__ newKey __13__ (c - 'a' + key) % 26;
            newKey __14__ 'a'__15__
            return __16__;
        } __17__ {
            //If c is not a correct char, don't encrypt.
            __18__
        __19__
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' _8_ 1 __9___ _10_ ___11___){
            _12_ newKey _13_ (c - 'a' + key) % 26;
            newKey _14_ 'a'_15_
            return __16__;
        } _17_ {
            //If c is not a correct char, don't encrypt.
            __18___
         _19_
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 __9___ __10__ ___11____){
            __12__ newKey __13__ (c - 'a' + key) % 26;
            newKey __14__ 'a'__15__
            return __16__;
        } __17__ {
            //If c is not a correct char, don't encrypt.
            ___18____
         __19__
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key  10   __ 11 __ ){
            12  newKey  13  (c - 'a' + key) % 26;
            newKey  14  'a' 15
            return  16  ;
        }  17  {
            //If c is not a correct char, don't encrypt.
            18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && __11__){
            __12__ newKey __13__ (c - 'a' + key) % 26;
            newKey __14__ 'a'__15__
            return __16__;
        } __17__ {
            //If c is not a correct char, don't encrypt.
            __18__
        __19__
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            __12__ newKey __13__ (c - 'a' + key) % 26;
            newKey __14__ 'a'__15__
            return __16__;
        } __17__ {
            //If c is not a correct char, don't encrypt.
            __18__
        __19__
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey  13  (c - 'a' + key) % 26;
            newKey  14  'a' 15
            return  16 ;
        }  17  {
            //If c is not a correct char, don't encrypt.
             18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey  14  'a' 15
            return   16  ;
        }  17  {
            //If c is not a correct char, don't encrypt.
              18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a' 15
            return   16   ;
        } 17  {
            //If c is not a correct char, don't encrypt.
              18
         19
        }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a';
            return   16   ;
        }  17  {
            //If c is not a correct char, don't encrypt.
              18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a';
            return newKey;
        } 17 {
            //If c is not a correct char, don't encrypt.
              18
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a';
            return newKey;
        } else {
            //If c is not a correct char, don't encrypt.
               18
         19
        }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a';
            return newKey;
        } else {
            //If c is not a correct char, don't encrypt.
            return c;
         19
    }
}
```

Fill in the Code

```
public class CaesarCypher{

    //Encrypts a character by taking an integer key in [1, 24]
    public static char encrypt(char c, int key){

        //Check if c is an ASCII character
        if ('a' <= c && c <= 'z' && 1 <= key && key <= 24){
            char newKey = (c - 'a' + key) % 26;
            newKey += 'a';
            return newKey;
        } else {
            //If c is not a correct char, don't encrypt.
            return c;
        }
    }
}
```

Can you guess what this code does?

Making Objects

- So far we've probably only used one object: Strings

Making Objects

- So far we've probably only used one object: Strings
- Objects are units composed of "behavior" (methods contained in the object), and "state" (variables contained in the object).

Making Objects

- So far we've probably only used one object: Strings
- Objects are units composed of "behavior" (methods contained in the object), and "state" (variables contained in the object).
- *We are objects*. We are each objects of the person class. We each have an age (int), a name (string), and a method called birthday (public void birthday(){age ++;}).

Making Objects

- So far we've probably only used one object: Strings
- Objects are units composed of "behavior" (methods contained in the object), and "state" (variables contained in the object).
- *We are objects*. We are each objects of the person class. We each have an age (int), a name (string), and a method called birthday (public void birthday(){age ++;}).
- To create an object, you must use the *new* keyword. The *new* keyword allocates memory in the computer for our object, and it can also instantiate variables within the object.

Making Objects

- To create an *instance* of an object:

```
Person Brian = new Person();
Brian.birthday();
System.out.println(Brian.age);
```

Constructors

- To initialize an object we can create a *constructor*.

Constructors

- To initialize an object we can create a *constructor*.
- Constructors for a `class <name>` look like:

```
public <name>(<optionalArguments>){...}
```

Constructors

- To initialize an object we can create a *constructor*.
- Constructors for a `class <name>` look like:

```
public <name>(<optionalArguments>){...}
```

```
public class Person{
    int age;
    public Person(){this.age = 0;}
```

Constructors

- To initialize an object we can create a *constructor*.
- Constructors for a `class <name>` look like:

```
public <name>(<optionalArguments>){...}
```

```
public class Person{
    int age;
    public Person(){this.age = 0;}
```

- Constructors are not necessary. Without one, an object will be created without setting or changing any of its variables using a *default constructor*.

Constructors and Overloading

- *this* is a key word that refers to the object that a method belongs to. It can be used to find variables and methods:

```
this.age
```

Constructors and Overloading

- *this* is a key word that refers to the object that a method belongs to. It can be used to find variables and methods:

```
this.age
```

- Objects can have more than one constructor as long as each one has different arguments. For example:

```
public class Person{
    int age;
    String firstname=""; String lastname = "";
    public Person(){this.age = 0;}
    public Person(int age){this.age = age;}
    public Person(int age, String firstname,
          String lastname){
        this.age = 0; this.firstname = firstname;
        this.lastname = lastname;}
```

So Why use Static?

- *Static* variables belong to a *class*, not an *instance* of a class.

So Why use Static?

- *Static* variables belong to a *class*, not an *instance* of a class.
- Static variables don't require a class to be instantiated, making them useful for constants.

How do we declare a constant?

So Why use Static?

- *Static* variables belong to a *class*, not an *instance* of a class.
- Static variables don't require a class to be instantiated, making them useful for constants.

How do we declare a constant?

- Static variables can also be used to hold information about a class.

How could we use static variables to keep track of the number of People?

So Why use Static?

- *Static* variables belong to a *class*, not an *instance* of a class.
- Static variables don't require a class to be instantiated, making them useful for constants.

  How do we declare a constant?
- Static variables can also be used to hold information about a class.

  How could we use static variables to keep track of the number of People?
- However, it is good coding practice to keep things in objects whenever possible. (Avoid over-using static.)

Creating an Object

```
public class Person{
    static int population = 0;
    int counter = 0;
    private final int ssn;

    public Person(int ssn){
       this.ssn = ssn;
       population++;
       counter ++;
    }

    public int getSSN(){
       return ssn;
    }
```

Instantiating an Object

An object can be instantiated in the `main` method:

```
public class PersonTest{
    public static void main(String[] args){
        Person p = new person();
        p.getSSN();
    }
}
```

Instantiating an Object

An object can be instantiated in the `main` method:

```
public class PersonTest{
    public static void main(String[] args){
        Person p = new person();
        p.getSSN();
    }
}
```

*Note: You can only use a default constructor when it's been declared `OR` the class has no constructors declared.

Objects and References

- In Java, variables can hold a *reference* to an object, a number that indicates where the object exists in memory.

Objects and References

- In Java, variables can hold a *reference* to an object, a number that indicates where the object exists in memory.
- They do not hold the object itself.

Objects and References

- In Java, variables can hold a *reference* to an object, a number that indicates where the object exists in memory.
- They do not hold the object itself.
- The == operator will compare the locations of two objects.
- The equals(Object o) method compares the contents of the objects.

Objects and References

- In Java, variables can hold a *reference* to an object, a number that indicates where the object exists in memory.
- They do not hold the object itself.
- The == operator will compare the locations of two objects.
- The equals(Object o) method compares the contents of the objects.
- Do you know of a class that uses the equals(Object o) method?

Objects and References

- In Java, variables can hold a *reference* to an object, a number that indicates where the object exists in memory.
- They do not hold the object itself.
- The == operator will compare the locations of two objects.
- The equals(Object o) method compares the contents of the objects.
- Do you know of a class that uses the equals(Object o) method?
- What does equals return?