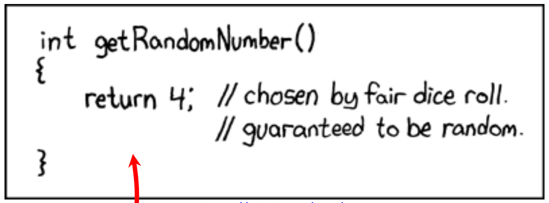
Static methods



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

http://xkcd.com/221/

Not actually a valid Java static method...

Programs thus far

- One big `main()`:

```
public class DiceRolling
{
   public static void main(String [] args)
   {
      int rolls  = 0;
      int sum    = 0;
      int target = (int) (Math.random() * 12) + 1;

      System.out.println("Rolling dice until I get " + target + ".");

      do
      {
         int dice1 = (int) (Math.random() * 6) + 1;
         int dice2 = (int) (Math.random() * 6) + 1;
         sum = dice1 + dice2;
         System.out.println(dice1 + " + " + dice2 + " = " + sum);
         rolls++;
      }
      while (sum != target);
      System.out.println("It took " + rolls + " rolls.");
   }
}
```

Programs thus far

- One big `main()`:

```
public class DiceRolling
{
   public static void main(String [] args)
   {
      int rolls  = 0;
      int sum    = 0;
      int target = (int) (Math.random() * 12) + 1

      System.out.println("Rolling dice until I ge

      do
      {
         int dice1 = (int) (Math.random() * 6) +
         int dice2 = (int) (Math.random() * 6) +
         sum = dice1 + dice2;
         System.out.println(dice1 + " + " + dice2
         rolls++;
      }
      while (sum != target);
      System.out.println("It took " + rolls + " r
   }
}
```

```
% java DiceRolling
Rolling dice until I get 4.
6 + 1 = 7
3 + 3 = 6
5 + 5 = 10
5 + 1 = 6
3 + 3 = 6
6 + 2 = 8
1 + 4 = 5
4 + 3 = 7
5 + 5 = 10
5 + 4 = 9
4 + 1 = 5
1 + 6 = 7
6 + 4 = 10
2 + 2 = 4
It took 14 rolls.
```

Programs thus far

- Problems with one big `main()`:
  - Doesn't scale to complex programs
  - Often find ourselves repeating similar code

```
public class DiceRolling
{
   public static void main(String [] args)
   {
      int rolls  = 0;
      int sum    = 0;
      int target = (int) (Math.random() * 11) + 2;

      System.out.println("Rolling dice until I get " + target

      do
      {
         int dice1 = (int) (Math.random() * 6) + 1;
         int dice2 = (int) (Math.random() * 6) + 1;
         sum = dice1 + dice2;
         ...
```

Using static methods

- Static methods
  - Already seen loads of "helper" methods:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num     = Integer.parseInt(args[0]);
double r    = Double.parseDouble(args[1]);
int x       = StdIn.readInt();
double rand = Math.random();
double v    = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Using static methods

- Static methods
  - Already seen loads of "helper" methods:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num     = Integer.parseInt(args[0]);
double r    = Double.parseDouble(args[1]);
int x       = StdIn.readInt();
double rand = Math.random();
double v    = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Some methods return a value.

Using static methods

- Static methods
  - Already seen loads of "helper" methods:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num     = Integer.parseInt(args[0]);
double r    = Double.parseDouble(args[1]);
int x       = StdIn.readInt();
double rand = Math.random();
double v    = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Some methods take a single parameter.

Using static methods

- Static methods
  - Already seen loads of "helper" methods:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num     = Integer.parseInt(args[0]);
double r    = Double.parseDouble(args[1]);
int x       = StdIn.readInt();
double rand = Math.random();
double v    = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Some methods take no parameters

Using static methods

- Static methods
  - Already seen loads of "helper" methods:

```
System.out.println("Hello world");
StdDraw.setPenColor(StdDraw.GRAY);
int num     = Integer.parseInt(args[0]);
double r    = Double.parseDouble(args[1]);
int x       = StdIn.readInt();
double rand = Math.random();
double v    = Math.pow(10.0, -2.3582);
StdDraw.setXscale(0.0, 10.0);
```

Some methods take two parameters.

Methods

- Methods:
  - Like a mathematical function
    - Given some inputs, produce an output value
  - Methods allows building modular programs
    - Reuse code, only invent the wheel once
  - When a method is called:
    - Control jumps to the method code
    - Argument passed to method copied to parameter variables used in method
    - Method executes and (optionally) returns a value
    - Execution returns to calling code

Flow of control

```
public class MethodJumping
{
    public static void printWorld()
    {
        System.out.print("world");
    }
    public static int addNums(int num1, int num2)
    {
        int result = num1;
        result = num1 + num2;
        return result;
    }
    public static void main(String [] args)
    {
        System.out.print("Hello");
        System.out.print(" ");
        printWorld();
        System.out.print(", 1 + 2 = ");
        int a = addNums(1, 2);
        System.out.println(a);
    }
}
```

```
% java MethodJumping
Hello world, 1 + 2 = 3
```

Anatomy of a method

- Goal: helper method than can draw a random integer between `start` and `end` (inclusive)

"Everybody can call me"

"I promise to return a value of this type"

"I need to know these things to do my job"

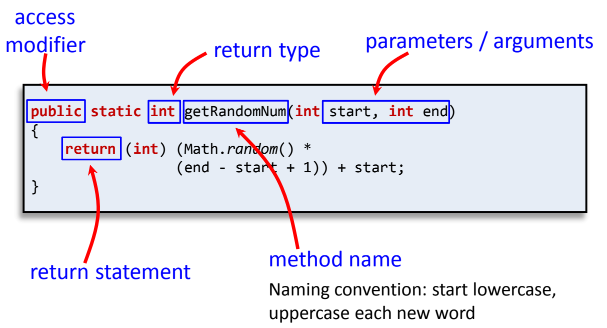
```
public static int getRandomNum(int start, int end)
{
    return (int) (Math.random() *
                 (end - start + 1)) + start;
}
```

"All done, end this method and return the result to whoever called me"

"The name I demand people use when they want my random goodness"

Terminology of a method

- Goal: helper method than can draw a random integer between `start` and `end` (inclusive)

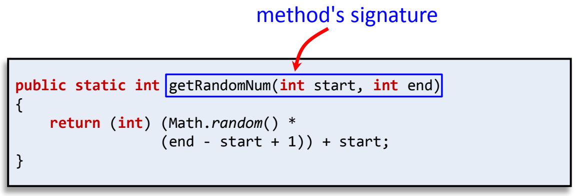


```
public static int getRandomNum(int start, int end)
{
   return (int) (Math.random() *
                (end - start + 1)) + start;
}
```

Naming convention: start lowercase, uppercase each new word

Method signature

- Signature: a method's name plus the number and type of its parameters
  - Note: does NOT include the return type!



```
public static int getRandomNum(int start, int end)
{
   return (int) (Math.random() *
                (end - start + 1)) + start;
}
```

Calling our new method

- Use handy new method in `DiceRolling`
  - Add somewhere inside `public class {}'s`

```
public class DiceRolling
{
   public static int getRandomNum(int start, int end)
   {
      return (int) (Math.random() *
                   (end - start + 1)) + start;
   }

   public static void main(String [] args)
   {
      int rolls  = 0;
      int sum    = 0;
      int target = getRandomNum(1, 12);

      System.out.println("Rolling dice until I get " + target + ".");
      do
      {
         int dice1 = getRandomNum(1, 6);
         int dice2 = getRandomNum(1, 6);
         sum = dice1 + dice2;
         ...
```

Calling our new method

- Alternative: put method in new class
  - Allows us to create a class with a bunch of helper methods (just like `StdIn.java, StdDraw.java`)

```
public class RandomUtil
{
   // Return random integer in [start, end] inclusive
   public static int getRandomNum(int start, int end)
   {
      return (int) (Math.random() *
                   (end - start + 1)) + start;
   }

   // Return random integer in [0, end] inclusive
   public static int getRandomNum(int end)
   {
      return (int) (Math.random() * (end + 1));
   }
}
```

`getRandomInt()` is ***overloaded:*** Two methods with same name, but different signatures (e.g. different number of parameters)

Using our new class

- Put `RandomUtil.java` in same directory
  - Methods qualified with `RandomUtil.` in front

```
public class DiceRolling
{
   public static void main(String [] args)
   {
      int rolls  = 0;
      int sum    = 0;
      int target = RandomUtil.getRandomNum(2, 12);

      System.out.println("Rolling dice until I get " + target + ".");
      do
      {
         int dice1 = RandomUtil.getRandomNum(1, 6);
         int dice2 = RandomUtil.getRandomNum(1, 6);
         sum = dice1 + dice2;
         System.out.println(dice1 + " + " + dice2 + " = " + sum);
         rolls++;
      }
      while (sum != target);
      System.out.println("It took " + rolls + " rolls.");
   }
}
```

17

A safer version

- Problem: What if caller sends in start > end?

```
public static int getRandomNum(int start, int end)
{
    return (int) (Math.random() *
                  (end - start + 1)) + start;
}
```

```
while (true)
   System.out.print(RandomUtil.getRandomNum(3, 1) + " ");
```

```
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 ...
```

18

A safer version

```
public static int getRandomNum(int start, int end)
{
    if (start < end)
       return (int) (Math.random() *
                    (end - start + 1)) + start;
    return (int) (Math.random() *
                 (start - end + 1)) + end;
}
```

As soon as a return is hit, method done

```
while (true)
   System.out.print(RandomUtil.getRandomNum(3,1) + " ");
```

```
3 1 1 1 2 2 3 1 1 1 3 2 3 1 2 1 3 3 2 2 2 2 1 3 1 3 1 3 3 3 1 3
2 1 2 3 1 2 2 3 2 1 1 3 2 2 2 1 3 2 2 2 3 3 1 1 1 3 3 3 1 3 2 1
3 3 1 3 3 3 3 1 1 2 1 1 3 1 1 3 1 1 2 2 2 2 2 1 2 3 2 2 3 3 3 3
2 1 2 2 ...
```

19

Pass by value

- Java passes parameters by value (by copy)
  - Changes to primitive type parameters do not persist after method returns
    - Primitive types: `int, double, char, long, boolean`

```
public static int sum(int a, int b)
{
   int result = a + b;
   a = 0;
   b = 0;
   return result;
}
```

```
int c = 2;
int d = 3;
System.out.println("sum = " + sum(c, d));
System.out.println("c = " + c);
System.out.println("d = " + d);
```

```
% java PassByVal
sum = 5
c = 2
d = 3
```

20

Pass by value, puzzler #2

```
public static int sum(int c, int d)
{
   int result = c + d;
   c = 0;
   d = 0;
   return result;
}
```

```
int c = 2;
int d = 3;
System.out.println("sum = " + sum(c, d));
System.out.println("c = " + c);
System.out.println("d = " + c);
```

```
% java PassByVal
sum = 5
c = 2
d = 3
```

Variables c & d in main program are not the same as c & d in sum()!

Array parameters

- Arrays can be passed as arguments

```
public class AverageArray
{
   public static double average(int [] nums)
   {
      long total = 0;
      for (int i = 0; i < nums.length; i++)
         total += nums[i];
      return (double) total / (double) nums.length;
   }

   public static void main(String [] args)
   {
      int [] vals = new int[1000];
      for (int i = 0; i < vals.length; i++)
         vals[i] = RandomUtil.getRandomNum(1, 10);
      System.out.println("avg " + average(vals));
   }
}
```

```
% java AverageArray
avg 5.508
```

Quiz: variable scope

What lines are the following variables in scope?

`nums`	**4-7**
`total`	**4-7**
`vals`	**12-15**
`i`	**5-6, 13-14**

```
00 public class AverageArray
01 {
02    public static double average(int [] nums)
03    {
04       long total = 0;
05       for (int i = 0; i < nums.length; i++)
06          total += nums[i];
07       return (double) total / (double) nums.length;
08    }
09
10    public static void main(String [] args)
11    {
12       int [] vals = new int[1000];
13       for (int i = 0; i < vals.length; i++)
14          vals[i] = RandomUtil.getRandomNum(1, 10);
15       System.out.println("avg " + average(vals));
16    }
17 }
```

Quiz: variable scope

What is the value of `total` printed at the end of `main()`?

`123`

What if we remove line 4?

`Compile error: total cannot be resolved to a variable`

```
00 public class AverageArray
01 {
02    public static double average(int [] nums)
03    {
04       long total = 0;
05       for (int i = 0; i < nums.length; i++)
06          total += nums[i];
07       return (double) total / (double) nums.length;
08    }
09
10    public static void main(String [] args)
11    {
12       long total = 123;
13       int [] vals = new int[1000];
14       for (int i = 0; i < vals.length; i++)
15          vals[i] = RandomUtil.getRandomNum(1, 10);
16       System.out.println("avg " + average(vals));
17       System.out.println("total " + total);
18    }
19 }
```

Added line

Added line

Quiz: variable scope

What if we remove line 12?

Compile error: total cannot be resolved to a variable

```
00 public class AverageArray
01 {
02    public static double average(int [] nums)
03    {
04       long total = 0;
05       for (int i = 0; i < nums.length; i++)
06          total += nums[i];
07       return (double) total / (double) nums.length;
08    }
09
10    public static void main(String [] args)
11    {
~~12       long total = 123;~~
13       int [] vals = new int[1000];
14       for (int i = 0; i < vals.length; i++)
15          vals[i] = RandomUtil.getRandomNum(1, 10);
16       System.out.println("avg " + average(vals));
17       System.out.println("total " + total);
18    }
19 }
```

Summary

- Static methods
  - Helper functions
    - Perform calculations
    - Output data
    - Consolidate similar code to one location
  - Methods have:
    - 0 or more input parameters
    - An (optional) return value
  - We're already experts at using them
    - `StdDraw.show(100), StdIn.readInt(), Math.abs()`
  - Now we can make our own methods!