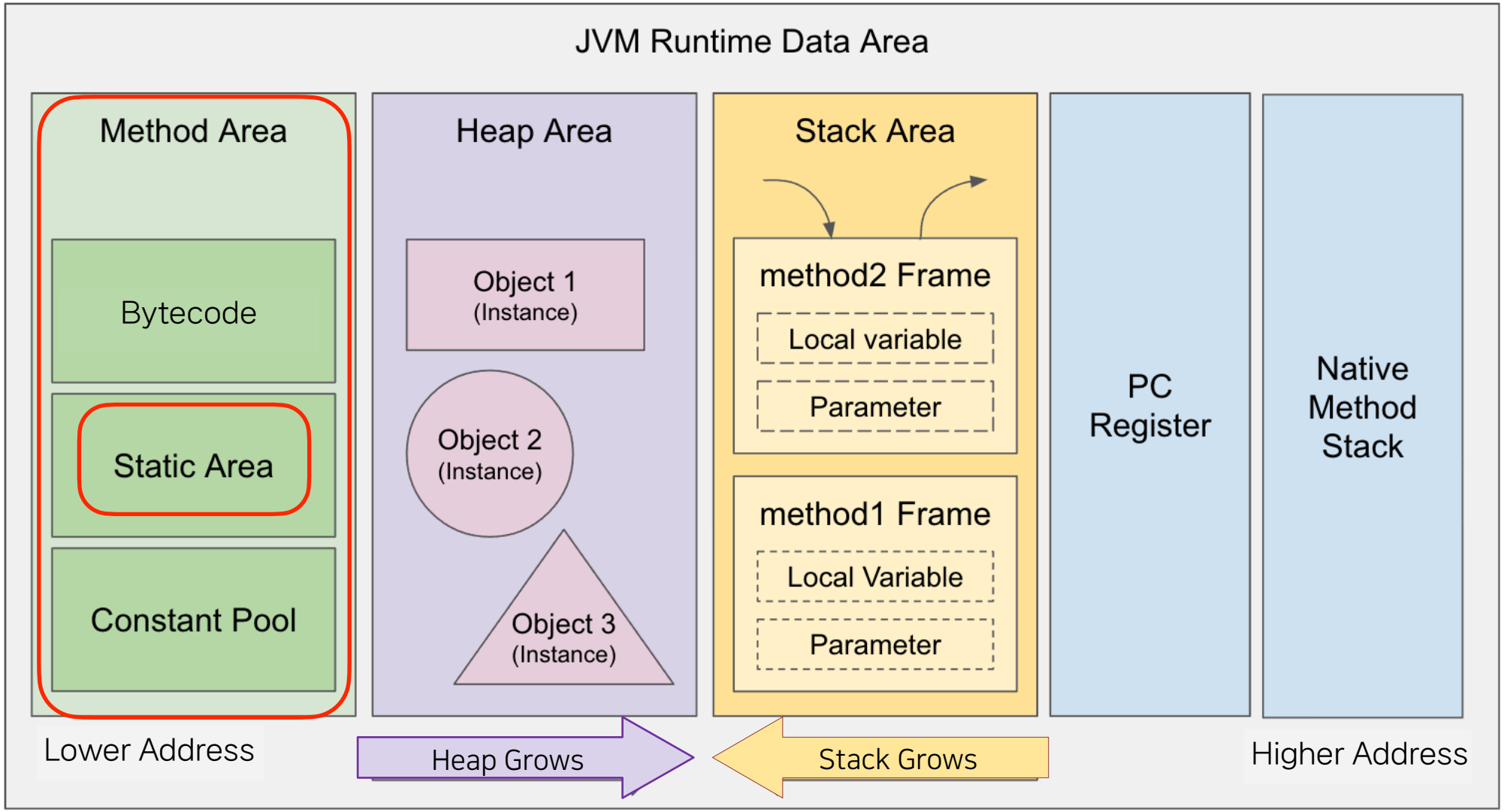
05_2 Static

Object-Oriented Programming

Instance Member vs Static Member

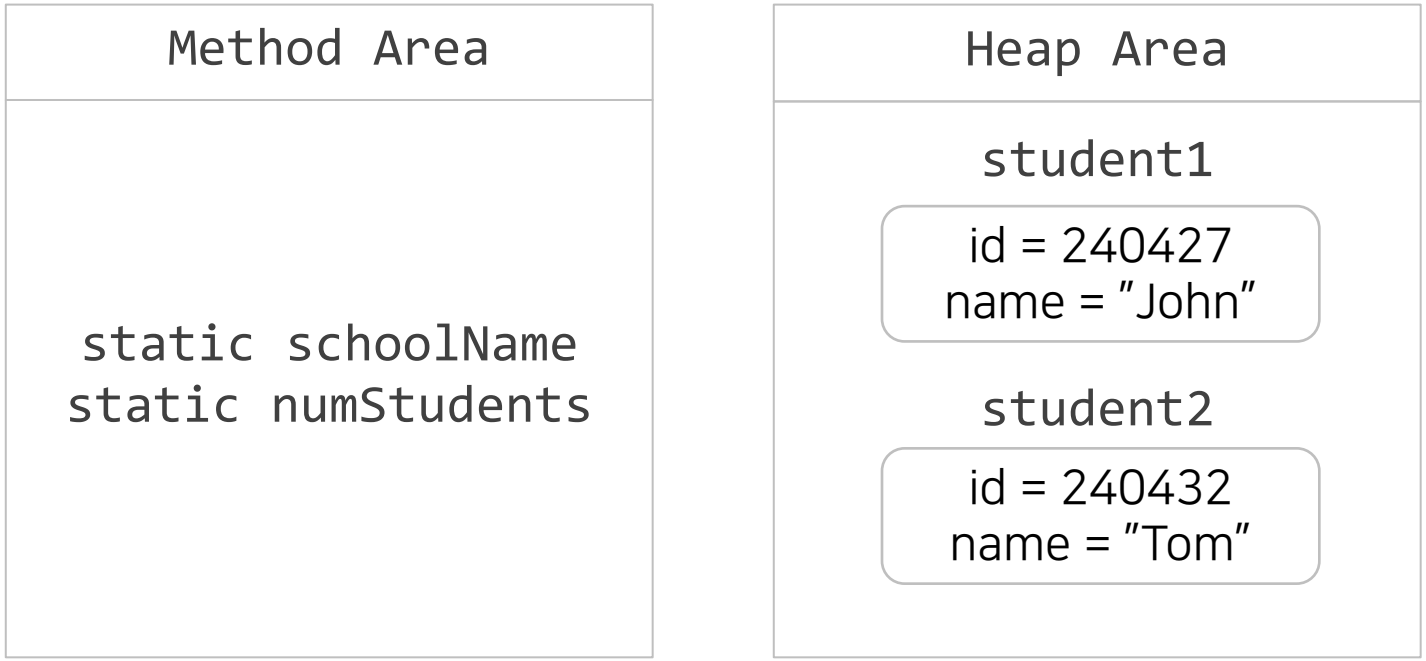
- Instance Member
  - Members that have separate values, one for each object
  - ex) Student class in a specific middle school
    - instance member: student's name
    - different students have different names
- Static Member
  - Exist only once in a class and are shared by all objects
  - ex) Student class in a specific middle school
    - static member: the name of the school
    - the same school's name for all students

Memory Structure of JVM (Revisited)



Instance and Static Member

```java
public class Student {
    private int id;
    private String name;
    public static schoolName;
    public final static numStudents = 120;
}
```



Static Variable – Without Object Creation

- Static variable can be accessed using only class name

- ex)

```
Student st1 = new Student();
Student st2 = new Student();
st1.setName(“John”);
st2.setname(“Tom”);
System.out.println(“st1’s name: “ + st1.getName());
Student.schoolName = “Saint Jone’s Middle School”; // static variable
```

The `Math` Class

- Provides a number of standard mathematical methods
  - In **`java.lang`** package, **no `import` needed**
  - All of its methods and data are **static**
  - Two predefined constants,
    - **`E`** (e: the base of the natural logarithm system)
    - **`PI`** (π = 3.141592 . . .)

ex)

```
area = Math.PI * radius * radius;   // πr²
double r = Math.random();
```

Math.random() Method

- Get 10 random integers in [1, 6]

```
System.out.println(Math.random()); // [0.0,1.0) ex)0.366755

for (int i = 0; i < 10; i++) {
    System.out.println((int)(Math.random() * 6) + 1);
}
```

```
2
6
6
5
5
2
3
3
3
2
```

$$0 \leq \text{Math.random()} < 1$$
$$0 \leq \text{Math.random()} * 6 < 6$$
$$1 \leq \text{Math.random()} * 6 + 1 < 7$$
$$(\text{int})(\text{Math.random()} * 6 + 1) \in \{1,2,3,4,5,6\}$$

Example: Math.Random() – CoinFlipDemo

```
public class CoinFlipDemo {

    public static void main(String[] args) {
        int counter = 1;
        while (counter <= 5){
            System.out.print("Flip number " + counter + ": ");
            int coinFlip = (int)(Math.random() * 2.0); // ∈ {0,1}
            if (coinFlip == 0)
                System.out.println("Heads");
            else // coinFlip == 1
                System.out.println("Tails");
            counter++;
        }
    }
}
```

```
Flip number 1: Heads
Flip number 2: Tails
Flip number 3: Tails
Flip number 4: Heads
Flip number 5: Tails
```

Random Object

```
import java.util.Random;

long seed = 365428;

Random rand = new Random();  // constructor
Random rand = new Random(seed);  // constructor with seed
// NOTE: when we give the same seed,
// the same random number series will be generated

int r = rand.nextInt(); // random integer [minimum int, maximum int]
r = rand.nextInt(n); // random integer in {0, 1, ..., n-1}
r = rand.nextInt(3) + 4;  // random integer in {4, 5, 6}
double rd = rand.nextDouble(); // random double 0.0 <= r < 1.0

nextBoolean() // random true or false
nextBytes()   // random byte integer
nextFloat()   // random float in [0.0, 1.0)
nextLong()    // random long integer
setSeed(long) // change the seed
```

Other Methods in Class Math (1/2)

- **public static double** pow(b, e)
  - ex) Math.pow(2.0, 3.0) returns $2^3 = 8.0$
- **public static int** abs(int), float abs(float), double abs(double), long abs(long)
  - ex) Math.abs(-6) returns 6, Math.abs(-5.5) returns 5.5
- **public static int** min(int,int), long min(long,long), float min(float,float), double min(double,double)
  - ex) Math.min(3, 2) returns 2
- **public static int** max(int,int), long max(int,int), float max(float,float), double max(double,double)
  - ex) Math.max(3,5) returns 5
- **public static int** round(float), long round(double)
  - ex) Math.round(3.4523) returns 3

Other Methods in Class Math (2/2)

- **public static double** ceil(double)
  - ex) Math.ceil(3.2), Math.ceil(3.8) both return 4.0
- **public static double** floor(double)
  - ex) Math.floor(3.2), Math.floor(3.8) both return 3.0
- **public static double** sqrt(double)
  - ex) Math.sqrt(4.0) returns 2.0

Wrapper Classes

- Class type corresponding to each of the primitive types
  - Helping class types to behave like primitive types
  - **Byte** for byte
  - **Short** for short
  - **Integer** for int
  - **Long** for long
  - **Float** for float
  - **Double** for double
  - **Character** for char
- Useful predefined constants and static methods

Boxing and Unboxing

- Boxing
  - Primitive type →wrapper class
  - auto boxing by assignment

- Unboxing
  - Wrapper class →primitive type
  - Using dedicated conversion method: …Value()

```
public class ATest5 {
    public static void main(String[] args)
{
        Byte bObj = 5;
        Short sObj = 15;
        Integer iObj = 256;
        Long lObj = 897584L;
        Float fObj = 243.563f;
        Double dObj = -98603.2543;
        Character cObj = 'y';

        byte b = bObj.byteValue();
        short s = sObj.shortValue();
        int i = iObj.intValue();
        long l = lObj.longValue();
        float f = fObj.floatValue();
        double d = dObj.doubleValue();
        char c = cObj.charValue();
    }
}
```

Automatic Unboxing

```
public class ATest5 {
    public static void main(String[] args) {
        Byte bObj = 5;
        Short sObj = 15;
        Integer iObj = 256;
        Long lObj = 897584L;
        Float fObj = 243.563f;
        Double dObj = -98603.2543;
        Character cObj = 'y';

        byte b = bObj;
        short s = sObj;
        int i = iObj;
        long l = lObj;
        float f = fObj;
        double d = dObj;
        char c = cObj;
    }
}
```

Constants in Wrapper Classes

- Min, Max values
  - `Integer.MAX_VALUE`, `Integer.MIN_VALUE`
  - `Double.MAX_VALUE`, `Double.MIN_VALUE`

- true and false
  - `Boolean.TRUE` and `Boolean.FALSE`

Some Static Methods in Wrapper Classes

- Conversion from String to other primitive types

```
int i = Integer.parseInt("365");
double d1 = Double.parseDouble("199.98");

String theString = "  673.23  “;
String trimedString = theString.trim();                  // “673.23”
double d2 = Double.parseDouble(trimedString);            // 673.23

String str = Double.toString(123.99);                    // "123.99"
```

Some Methods in Class Character (1/2)

- public static char toUpperCase(char arg)
  - ex) char c = Character.toUpperCase('a'); // 'A"
- public static toLowerCase(char arg)
  - ex) char c = Character.toLowerCase('A'); // 'a'
- public static boolean isLowerCase(char arg);
  - ex) boolean answer = Character.isLowerCase('c'); // true
- public static boolean isWhiteSpace(char arg);
  - ex) boolean answer = Character.isWhiteSpace('\t'); // true
  - // white space characters: space (blank), tab (\t), line break (\n)
- public static boolean isLetter(char arg);
  - ex) // letter: alphabet character: a ~ z, A ~ Z
  - // non-letter character: %, &, ^, *, ...

Some Methods in Class Character (2/2)

- public static boolean isDigit(char arg)
  - ex) // digit: number character such as ‘0’ ... ‘9’
- public static boolean isLetterOrDigit(char arg)
  - ex) // ‘a’...’z’, ‘A’...’Z’, ‘0’...’9’

Invocation Counter (1/2)

```
// Counting how many invocation made for all methods
// using static variable

public class InvocationCounter
{
    private static int numberOfInvocations = 0;

    public void demoMethod( ) {
        numberOfInvocations++;
    }

    public void outPutCount( ) {
        numberOfInvocations++;
        System.out.println("Number of invocations so far = "
        + numberOfInvocations);
    }
```

Invocation Counter (2/2)

```
    public static int numberSoFar( ) {
        numberOfInvocations++;
        return numberOfInvocations;
    }

    public static void main(String[] args) {
        InvocationCounter object1 = new InvocationCounter( );
        object1.outPutCount( );  // 1
        for (int i = 1; i <= 5 ; i++)
            object1.demoMethod( );  // +1
        object1.outPutCount( ); // 7

        InvocationCounter object2 = new InvocationCounter( );
        for (int i = 1; i <= 5 ; i++) {
            object2.demoMethod( ); // +1
            object2.outPutCount( ); // +1
        }

        System.out.println("Total number of invocations = " + numberSoFar( )); // 18
    }
}
```