Programming Concepts Using Java

Quiz 2 Revision

W08:L01: Cloning

- What if we want separate but identical objects
- Object defines a method `clone()`. However, it does shallow copy.
- Deep copy recursively clones nested objects
- To allow `clone()` to be used, a class has to implement `Cloneable` interface
- `clone()` is protected by default. Override as `public` if needed
- `clone()` in `Object` throws `CloneNotSupportedException`, catch or report this exception

W08:L02: Type inference

- Automatic type inference can avoid redundancy in declarations

  `Employee e = new Employee(...)`

- Java allows limited type inference
  - Only for local variables in functions
  - Not for instance variables of a class
- Challenge is to do this statically, at compile-time
- Use generic `var` to declare variables
  - Must be initialized when declared
  - Type is inferred from initial value
- Be careful about format for numeric constants
- For classes, infer most constrained type
  - `e` is inferred to be `Manager`
  - `Manager` extends `Employee`
  - If `e` should be `Employee`, declare explicitly

```
var b = false; // boolean

var s = "Hello, world"; // String

var d = 2.0; // double

var f = 3.141f; // float

var e = new Manager(...); // Manager
```

W08:L03: Higher order functions

- Passing a function as an argument to another function
- In object-oriented programming, this is achieved using interfaces – encapsulate the function to be passed as an object
- Lambda expressions denote anonymous functions
  - `(Parameters) -> Body`
  - Return value and type are implicit
- Interfaces that define a single function are called functional interfaces
  - `Comparator`, `Timerowner`
- Substitute wherever a functional interface is specified

```
String[] strarr = new ...;
Arrays.sort(strarr, (String s1, String s2) -> s1.length() - s2.length());
```

- Limited type inference is also possible
  - Java infers `s1` and `s2` are `String`

```
String[] strarr = new ...;
Arrays.sort(strarr, (s1, s2) -> s1.length() - s2.length());
```

- More complicated function body can be defined as a block

W08:L03: Higher order functions

- If the lambda expression consists of a single function call, we can pass that function by name – method reference
- We saw an example with adding entries to a Map object – here sum is a static method in Integer

```
Map<String, Integer> scores = ...;
scores.merge(bat,newscore,Integer::sum);
```

- Here is the corresponding expression, assuming type inference

```
(i,j) -> Integer::sum(i,j)
```

- `ClassName::StaticMethod` – method reference is `C::f`, and corresponding expression with as many arguments as `f` has

```
(x1,x2,..,xk) -> C::f(x1,x2,...,xk)
```

- `ClassName::InstanceMethod` – method reference is `C::f`, and called with respect to an object that becomes implicit parameter

```
(o,x1,x2,...,xk) -> o.f(x1,x2,...,xk)
```

- `object::InstanceMethod` – method reference is `o::f`, and arguments are passed to `o.f`

```
(x1,x2,..,xk) -> o.f(x1,x2,...,xk)
```

W08:L04: Streams

- We can view a collection as a stream of elements
- Process the stream rather than use an iterator
- Declarative way of computing over collections
- Create a stream, transform it, reduce it to a result
- Processing can be parallelized
  - `filter()` and `count()` in parallel
- Apply `stream()` to a collection
  - Part of Collections interface
- Use static method `Stream.of()` for arrays
- Create a stream, transform it, reduce it

```
long count = words.stream()
             .filter(w -> w.length() > 10)
             .count();
}

long count = words.parallelStream()
             .filter(w -> w.length() > 10)
             .count();
}
```

W08:L04: Streams (Cont.)

- Static method `Stream.generate()` generates a stream from a function
- `Stream.iterate()` — a stream of dependent values
- `filter()` to select elements – takes a predicate as argument
- `map()` applies a function to each element in the stream
- `flatMap()` flattens (collapses) nested list into a single stream
- Make a stream finite — `limit(n)`
- Skip n elements — `skip(n)`
- Stops when element outmatches a criterion — `takeWhile()`
- Start after element outmatches a criterion — `dropWhile()`
- Number of elements — `count()`
- Largest and smallest values seen - `max()` and `min()`
- First element — `findFirst()`
- What happens if the stream is empty? Return value is optional type